# 

**School of Physical Sciences**

**Master’s thesis submitted in partial   
fulfilment of requirements for M.Sc. IT Security**

**ANALYSIS OF ANTI-VIRTUAL MACHINE METHODS USED BY MODERN MALWARES.**

**By**

**CHIDIADI NWAOGU**

**PROJECT SUPERVISOR: MARTIN BATEMAN**

**Date of submission: 13th of August 2021  
  
Word count: 8019**

# ACKNOWLEDGEMENT

I want to use this medium to thank the school and my supervisor who gave me a chance and prepared me for this project. I also extend my thanks to my parents who made my master’s programme possible through their unfailing support.

# ABSTRACT

This study observed how different malware behaved in virtual machines in a bid to understand how malwares detect virtual machines. The study consisted of setting up virtual machines with a slight difference in their configurations aimed to bypass the malware’s detection techniques. The study was directed by four specific objectives which included to: determine which malware use virtual machine (VM) detection techniques, determine the techniques these malwares use in detecting virtual machines, use obfuscating methods to configure virtual machines in which malwares will run as they would on a real machine, find out the usefulness of obfuscation as a countermeasure technique. The study was further guided by three formulated research questions. In carrying out the experiment, one PC was used to remotely connect to VMs hosted on virtual private services. The specification of the PC included a hard drive size of 256GB, 16 GB RAM running on MacOS. Upon execution of a malware, its activities were recorded. The computer was restored to the created restore point prior to each execution. From the results, there was no significant utilisation of anti-virtual machine techniques in modern malwares as just three out of the twenty malwares tested exhibited virtual machine detection.

Table of Contents

[ACKNOWLEDGEMENT 2](#_Toc79742219)

[ABSTRACT 3](#_Toc79742220)

[CHAPTER ONE 6](#_Toc79742221)

[INTRODUCTION 6](#_Toc79742222)

[1.1 Background to the Study 6](#_Toc79742223)

[1.2 Statement of the Problem 9](#_Toc79742224)

[1.3 Aim and Objectives of the Study 9](#_Toc79742225)

[1.4 Research Questions 10](#_Toc79742226)

[1.5 Significance of the Study 10](#_Toc79742227)

[1.6 Scope of the Study 11](#_Toc79742228)

[CHAPTER TWO 12](#_Toc79742229)

[LITERATURE REVIEW 12](#_Toc79742230)

[2.1 Malware Types 12](#_Toc79742231)

[2.1.1 Virus 12](#_Toc79742232)

[2.1.2 Worm 14](#_Toc79742233)

[2.2 Infection Techniques 14](#_Toc79742234)

[2.2.1 Backdoor 15](#_Toc79742235)

[2.2.2 Trojan horse 15](#_Toc79742236)

[2.2.3 Rootkit 16](#_Toc79742237)

[2.2.4 Spyware 16](#_Toc79742238)

[2.2.5 Adware 17](#_Toc79742239)

[2.3 Detection Techniques 17](#_Toc79742240)

[2.3.1 Static investigation 17](#_Toc79742241)

[2.3.2 Dynamic investigation 18](#_Toc79742242)

[2.4 Overview of existing dynamic examination procedures 19](#_Toc79742243)

[2.4.1 Investigation in emulator 21](#_Toc79742244)

[2.4.2 Analysis in a virtual machine 22](#_Toc79742245)

[2.5 Limitations of dynamic examination 23](#_Toc79742246)

[CHAPTER THREE 25](#_Toc79742247)

[METHODOLOGY 25](#_Toc79742248)

[3.1 Introduction 25](#_Toc79742249)

[3.2 Virtual Machine Technology 25](#_Toc79742250)

[3.2.1 Sandbox Approach 25](#_Toc79742251)

[3.2.2 Hyper V manager 26](#_Toc79742252)

[3.2.3 Oracle virtual box 27](#_Toc79742253)

[3.3 Malwares Virtual Machines Detection Techniques 28](#_Toc79742254)

[3.3.1 Hardware fingerprinting 28](#_Toc79742255)

[3.3.2 Registry check 28](#_Toc79742256)

[3.3.3 Memory check 30](#_Toc79742257)

[3.3.4 VMware communication channel check 31](#_Toc79742258)

[3.3.5 File and process check 32](#_Toc79742259)

[CHAPTER FOUR 34](#_Toc79742260)

[PROCEDURES AND PRECAUTION 34](#_Toc79742261)

[4.1 ETHICS CONSIDERATION 34](#_Toc79742262)

[4.2 MATERIALS 35](#_Toc79742263)

[4.3 STEPS TAKEN 35](#_Toc79742264)

[4.4 RESULTS AND ANALYSIS 36](#_Toc79742265)

[4.5 LIMITATIONS TO STUDY 37](#_Toc79742266)

[4.6 STUDY DISCUSSION 37](#_Toc79742267)

[CONCLUSION 39](#_Toc79742268)

[REFERENCES 40](#_Toc79742269)

[APPENDICES 44](#_Toc79742270)

## CHAPTER ONE

# INTRODUCTION

## 1.1 Background to the Study

With the recent innovations drastically and almost irreversibly influencing our everyday lives over the course of recent years, people have turned so dependent and glued to the same stuff that is meant to liberate us. Specifically, as the Internet turns out to be increasingly universal all throughout the planet, the digital threats have additionally gotten increasingly common and genuine. The absence of satisfactory insurance systems and frameworks on the regular user’s PC and obliviousness and downplay about security dangers have caused digital criminals to dispatch security threats.

In no distant time, with Internet Protocol version 6 [IPv6] (Schipka, 2009; Rid and McBurney, 2012) and network address translation (NAT), practically all gadgets including vehicles, stoves, child monitors, television sets, fridges, and so on will be linked with an IP address and will be directed or controlled remotely. Thus, people will become soft targets to cyber-attacks. Tragically, since these electronic gadgets (also referred to as Internet of Things) have hardly been configured with sufficient security as part of its design, people will experience lethal outcomes when these gadgets get undermined by cyber attackers. For example, someone can imagine the situations where somebody hacks a vehicle and afterward finds an approach to inactivate the vehicle's brake system, or a scenario where a cyber attacker compromises an oven situated in an apartment and then, fire hazard can be vindictively initiated.

As a matter of fact, this few situations illustrate how huge the field of the internet is and how genuine result can be anticipated from an attack. When carrying out a digital campaign, the most well-known and successful way utilized by attackers is to exploit malware. Malware, which is malevolent software, generally alludes to any type of threatening software intended for different objectives like stealing individual data (for example charge card information, client accounts, email records), utilising it as an access for attacking different hosts, carrying out Distributed Denial of Service (DDoS), and so on, without the owner’s permission. For instance, worms commonly disseminate via the exploitation of server-side weaknesses over the cyber space. When a target has been attacked or accessed, a malware can place extra payload to control it remotely. Thus, the compromised system turns into part of a larger network, which is also referred to as a botnet in malware domain. In the internet, botnets are broadly utilized in launching DDoS hacks, distributing phishing messages, hosting weaknesses to abuse client-side systems, and so on.

The malware domain is turning into a business sector of digital weapons by maximising the rapidly changing features of malware and discovering 0-day exploits (attacks for unpatched weaknesses) (Alexandr et al., 2011; Flanagan, 2011). The present malware has advanced from crude and replicating viruses that alter OS systems and obliterate client files to profoundly hesitant and adaptable bits of software that permit digital attackers to dispatch ever-increasingly complex and designated attacks. Essentially, the malware authors are money oriented, and their key targets are companies. Indeed, some states/areas create malware in quest to uphold their political, conciliatory, and military strategies.

Sadly, despite the fact that malware creation and intricacy drastically increased, the information needed by an attacker to send malware has significantly diminished in the course of the last tens of years. This connection between the threat and cash investment obviously shows the damaging effect of studies on malware creation targeting huge size of computer systems. The present circumstance comes from the elevating utilization of easy to understand and automated malware development systems, like Metasploit. With these attacks packs, digital criminals can undoubtedly and easily (with minimum human intervention) dispatch attacks that infect systems in their quest to play out their own vindictive objectives.

A portion of these toolboxes are free, open source, whereas a few of them are marketed on the Internet's fraud market. Furthermore, as the way toward developing a malware test without any preparation is an exceptionally intricate and time consuming task and needs impressive capabilities, malware creators utilize runtime packers and obfuscation principles (O’Kane et al., 2011; You and Yim, 2010; Sahay and Sharma, 2014) (polymorphism and metamorphism). This prompts the hazardous elevation in malware types, which are typically indistinguishable however statically varying samples. As per Cisco, approximately four hundred thousand malware types have been detected daily during 2014 (Cisco, 2014). It is notable that most of these malware is the variant of the recently known samples.

Then again, identifying the group of a malware saves the re-analysis of sample instances and empowers scholars to zero in on new or concealed malware instances. Consequently, the categorization of malware samples into suitable classes is just about as significant as malware detection.

The current examination will observe how different malware act in virtual machines in a bid to see how malwares detect virtual machines. Virtual machine has been an imperative device for studying how malwares act since it is a protected environment. Notwithstanding, malwares have gotten more modern and complex since its inception in the beginning days of the internet, today more up to date malwares can detect virtual machines making it hard to notice these malwares in a virtual machine.

## 1.2 Statement of the Problem

Right now, cutting edge malware detection depends on static malware detection which comprises of analyzing documents without executing them and by and large using a marked database or rule-set to work. Albeit these arrangements are compelling and quick for known malware, they often have low detection rates on obscure instances and variations of older known samples. In the literature, scholars by and large utilize scoring systems and personal conduct standards to survey destructiveness. Notwithstanding, there is not sufficient studies in the field of the automatic grouping of malware and similarity of malware families as a function of their feature. On another viewpoint, the proposed techniques experience the ill effects of the scourge of dimensionality, which restricts the feature space to keep away from blast of searching intricacy level. Then again, while malware behaviours develop relatively with the quantity of activities of a given sample, the codes require longer execution time and cause the analysis to be inefficient.

To adapt to that issue and to give more digitization in the arms contest between malware creators and experts, there is a solid requirement for novel and versatile malware analysis environments to consequently investigate huge quantities of malware samples progressively.

This study will analyse prevalence of virtual environment detection techniques usage in malwares using data from own experiment combined with existing and verifiable tests or experimentation results relevant to the research topic. The researcher will subsequently determine the popularity of these techniques amongst malware which is relevant to inform the countermeasures against malware attacks.

## 1.3 Aim and Objectives of the Study

These are the aims of these paper are as follows

1. Analysis and interpretation of data obtained from experimenting with large malware sample set.
2. Determine the extent at which VM evasion techniques are used in contemporary malware.

## 1.4 Research Questions

The following research questions have been formulated to guide the study:

1. What techniques do malwares employ to detect virtual machines?
2. What are the characteristics of these anti VM behaviours?
3. What are the differences between old malwares and contemporary malwares in how they detect malware? Is it an old practice or new one?

## 1.5 Significance of the Study

Today, digital world is being advanced by a huge assortment of computerized information and innovation-based services. An increasing pace of far off and portable utilization prompts a surprising reliance on information security. Analysis and detection of malignant software is a challenging errand because of the introduction of cutting-edge obfuscation methods by malware creators. In studying malware, quite possibly the main initial steps are comprehending what malware is, what types exist, and how we can approach identifying and relating malware. Virtual machine has been a vital tool for studying how malwares behave because it is a safe environment. However, malwares have become more sophisticated since its inception in the early days of the internet that as such newer malwares can detect virtual machines making it difficult to observe these malwares in a virtual machine. This project investigates this issue in a bid to answer the research questions formulated to guide the study. The study will ascertain which malware use VM detection techniques and also ascertain the techniques these malwares use in detecting virtual machines.

## 1.6 Scope of the Study

The study will consist of setting up virtual machines with a slight difference in their configurations aimed to bypass the malware’s detection techniques. The data to be obtained will be in qualitative form. The data set will record; each virtual machine, if a particular malware ran on each of the virtual machine, the path or directory in which the malware stores itself, if the saved passwords on the virtual machines are seen by each malware.

# CHAPTER TWO LITERATURE REVIEW

## 2.1 Malware and virtual machines

It is well documented that some malware can differentiate between when they are being run on a virtual machine from when they are in a real user's computer. To understand this concept, there is a need for the reader to understand virtualisation technology. According to Kusnetzky, virtualisation is a way to abstract applications and their underlying components away from the hardware supporting them and present a logical or virtual view of these resources (Kusnetzky, 2011). While this definition is true, it is well to note that virtual view may be different from the physical view. The relationship between the is further explain using the Kusnetzky group model of virtualisation.

Graphical user interface, application

Description automatically generated

In the above model, Kusnetzky shows the different layers of technology in which virtualisation occurs and

VMs being emulations of physical machines, make use of VMM to provide hardware administrative assistance to the VM user. These VMMs allow the VM connect or use the host resources and peripherals to operate. The Vms will often make use of additional or custom programs called guest programs to facilitate communication between itself and the host computer. This means that VMs, rely on the functionality of a host machine to function optimal. Although VMs can perform most operations as real computers do, they cannot be 100% identical to real hardware because of the underlying dependencies on a real machine. Some malwares have been observed to capitalise on these differences between Vms and real hardwares to evade VM. Some of these differences facilitate the detection by malwares.

## 2.2 Detection Techniques

Effective location of malware files is viewed as the initial move toward examination of dangerous programs. These approaches are presented by static and dynamic examination.

### 2.2.1 Static investigation

Static investigation comprises of analyzing a binary file without executing it on the system. Since the source program is not accessible, malware is dismantled, and the developed execution ways are examined. Potential functionalities are assessed, basic network signatures are generated based on the accumulated data, lastly, probability of being dangerous is ascertained. The initial step of static investigation comprises of searching for clear markers of dangerous programs. This process is like the examination method of conventional anti-virus where file fingerprint (normally file hashes, e.g., MD5, SHA) is determined and coordinated with a definitely known malware. The subsequent process is the profound file examination where the file format and content are examined. The accompanying checks are implemented:

File unloading: Packing of the file ought to be ascertained. In the event that the file is packed prior to profound search, it must be unloaded and unadulterated executable needs acquired. There are diverse packing techniques to obfuscate a malware, yet unpacking methodologies are profoundly delicate to its previous packer, and getting unadulterated executable program is quite difficult. Be that as it may, specialists are researching on developing common unpacker for various packers (Bohne, 2008; Choo et al., 2010).

Plain text matching: The plain text of the executable is assessed, and data about being dangerous is assembled as much as could be feasible. By and large, strings utility is utilized to investigate plain text in the file.

Dismantling: Disassembly is utilized to inspect the machine program of the executable file. Checking careful and steadily the machine code, as in the debugger case, assists with noticing the program’s steps. This is a high level malware identification strategy, yet it may not uncover all details of the dangerous software on the grounds that it is not being completely executed.

Static malware investigation is more or less direct and quick. It guarantees security and wellbeing of the PC/systems due to its capacity of identifying dangerous file without execution. In any case, a compromise exists between its effortlessness and reliability; static identification mechanism can be incapable against complex malware and may miss significant dangerous behaviours (Moser et al, 2007).

### 2.2.2 Dynamic investigation

Dynamic investigation strategies entails executing the malware and noticing its conduct on the system. Ordinarily, malwares have capacities to vary a few kinds of things on the compromised gadget, so dynamic examination comprises of checking the accompanying behaviours:

Unpredictable memory: Malware can flood buffers and utilize the unwanted memory areas to access the gadget. By catching and examining the gadget memory, it is feasible to decide if and how the malware utilizes the memory.

Registry/design variations: Variations in the registry might be a proof toward dynamic investigation. Malwares frequently alter registry values to acquire persevering admittance to the system.

File activity: Malware may likewise add, change, or erase the files. So by checking document file activities important data about the malicious conduct can be acquired.

Processes/services: Malware may cripple anti-virus engines to satisfy their functions, leap to different processes to deter examination, or put in new services to acquire diligent admittance to the system.

Network connection: Monitoring the network connections is the fundamental piece of dynamic examination to locate the malware’s presence. Destination IP address, port number, and protocol can be examined or accessed to distinguish malware’s interaction with the command and-control (C&C) server.

Monitoring of these behaviours has the capacity to bring about significant data about software’s target, which is hard to be accumulated by other identification strategies. However, a few restrictions in detection may happen due to the evolving of malware features versus anti-dynamic investigation, for example, anti-VM, anti-debugger, and significant functionalities may not be energized for some sort of malwares.

## 2.3 Overview of existing dynamic examination procedures

Dynamic malware examination is an ongoing research area. Some techniques exist for examining dangerous programs with minimal human intervention. These can be assembled into three groups (Ligh et al., 2010):

Difference-based: Examine contrasts between two snapshots of the system, one prior to the malware execution, and the other one subsequent to its execution.

Notification-based: Use of notification means for the operating system calls triggered by specific events, for instance, registry key variations, file/folder modifications.

Hook-based: Hook application programming interfaces (APIs) in user mode or kernel mode to assess changes implemented on the system.

Most dynamic malware investigation instruments, for example, Anubis (Anubis, 2021), CWSandbox (Willems et al, 2007), and Norman Sandbox (Norman, 2021) track run-time activities carried out by malwares. Notwithstanding, Regshot (Redshot, 2021), which is utilized to identify file and registry variations, tracks these variations by taking snapshots of the delicate operating system parts (e.g., files, folder, registry keys, etc). Security mechanism of majority of sandboxes (example CWSandbox, Anubis, and Cuckoo Sandbox (Cuckoobox 2021), which are utilized for executing untrusted or dangerous codes in a protected environment without compromising real systems, present genuinely comparative ways to uncover the behaviour of the malware. The normal assignment of these tools incorporates an assortment of insights regarding the malware, for example, network activities and developed files during its execution time. These events can be observed via their user or kernel space capacities. Investigating dangerous programs in user space assists with getting significant level-involved capacities, for example, listing the active processes, discovering locked files, and monitoring network connections. Yet, when working in user space, hidden process or connections, which are installed in the kernel space, cannot be identified. To get to data hidden in the operating system, examination tools must have a kernel space module. These modules are required to utilize kernel space capacities to assemble data hidden from the user. In any case, kernel space investigation needs profound and strong information on Windows operating system (OS), and it is quite a challenging task. As an option to kernel space investigation, the process of interrupting function calls is an often utilized strategy to monitor behaviours of the malware and flow of the executable logic. Most dynamic analysis systems, for example, CWSandbox, BitBlaze (Song et al., 2008), and TTANalyze (Bayer et al., 2006) (presently known as Anubis) utilize this method, which is also referred to API hooking. The idea of hooking is straightforward: the call made by a system to a function can be diverted to a specially characterized function. API hooking must be straightforward and imperceptible by the malware. On the off chance that a malware identifies API hooking, it might alter its practices or functions so as to be covered up.

From the side of malware, API-hooking trap might be prevented by calling undocumented kernel functions straightforwardly as opposed to utilizing API functions. Be that as it may, in the event of utilizing kernel functions, the objective set to be contaminated by malware will be restricted by the systems whose version of the OS and service patch level is determined. This circumstance may develop contradictions with the overall aim of numerous malwares about infecting huge measure of systems.

Dynamic investigation instruments can be classified under two classes distinguished by execution platform: examination in unadulterated programming emulator and investigation in virtual machines. Dynamic Binary Instrumentation (DBI) is a unique examination strategy that permits self-assertive code to be executed when a program is running. DBI structures have begun to be utilized to dissect pernicious applications. Therefore, various methodologies have converged to identify and stay away from them. Usually alluded to as parted character malware or equivocal malware are bits of malignant programming that consolidate pieces of code to recognize when they are under DBI system examination and in this manner emulate harmless conduct. Late investigations have scrutinized the utilization of DBI in malware examination, contending that it builds the assault surface (Ailton Dos Santos Fh, Ricardo J. Rodríguez and Eduardo L. Feitosa; Online:13 August 2021).

### 2.4.1 Investigation in emulator

An emulator is a piece of program that mimics the equipment. A software emulator does not implement code straightforwardly on the underlying hardware. Rather, instructions are captured by the emulator, transformed to a relating set of guidelines for the target platform, and lastly, implemented on the hardware. For instance, Qemu ( Qemu, 2021) is an open source and full system emulator where the processor and peripherals are imitated by a program. Anubis, Renovo (Kang et al., 2007), and Hookfinder (Yin et al., 2008) utilizes the Qemu emulator to examine malwares. Notwithstanding, emulator method is definitely not a straight-forward answer for identifying malware. As expressed in (Ferrie, 2020), malwares can distinguish the phase of emulation. For instance, identifying imperfect emulation of the CPU or implementation period of the particular commands permits a malware sample to perceive the circumstance about executing in an emulator.

### 2.4.2 Analysis in a virtual machine

Malware analysis is key in countering potent threat on the Web. In powerful investigation, a malware test is executed in a controlled environment and its activities are logged. Through powerful investigation, an examiner can rapidly get an outline of malware conduct and can choose whether or not to enjoy into drawn-out manual examination of the example. Be that as it may, normal unique investigation opens the Web to the dangers of an executed malware (like portscans) in light of the fact that best in class disguise procedures of malware frequently require full Web access. For instance, a missing connection to the Web or the inaccessibility of a particular server regularly causes the malware to not trigger its noxious conduct. (Gorecki, Freiling and Holz, 2011).

Malwares are recognized based on their practices and characteristics by applying precautions with respect to virtual machine identification strategies. The primary advantage of the method is the identification of malwares as a function of their practices in the virtual environment without being detected by malwares.

Nowadays, both system administrators and users lean toward VMs in light of the fact that they are not difficult in modifying a machine from a preview. Following these inclinations, malware creators understood that virtualization technique is utilized to analyse malicious executables, and they began to obfuscate their source programs with VM deceits. With these methods, a malware tries to identify if it is being run within a VM or on a real machine. In the event that the VM is identified, it can act variedly or basically does not run. Problems with dependability and inadequate test data might deceive the analyst. Virtualization entails simulating portions of a PC’s hardware, while most tasks actually happen on the real equipment for efficiency sakes. In this way, virtualization is clearly quicker as regards emulation. VMs additionally give devoted assets like emulators. A snapshot of VM assets (for example, virtual mass storage, CPU, and memory contents) can be recorded for reestablishing reasons. This characteristics can be utilized to decrease the time needed to examine a malware since installation is not needed to develop again a spotless instance of the examination environment. For instance, CWSandbox utilizes this method to assess a malware sample in a virtual Windows environment. This component comprises on running projects in a different/detached climate by controlling every one of the assets designated if there should be an occurrence of harm. This is viewed as a suitable conflict component against obscurity.

There are a few apparatuses that utilization this method. Such apparatuses can emulate malware collaboration just as getting and archiving changes made to a tainted framework. When an example of malware is in execution, it is critical to gather however much data as could be expected to decide changes acted in the framework, recognize designs, and comprehend its conduct.

Presently, this method could be outmaneuvered as malware engineers are including directions for distinguishing a sandboxed climate to forestall malware execution. (Barriga and Yoo, 2017)

## 2.5 Limitations of dynamic examination

Due to the advancing attributes of malwares, techniques dependent on dynamic investigation may present drawbacks in assessing dependability. Malware sandboxes do not consider any command-line alternatives; that is, they implement the executable and observe the behaviours. Examination of candidate and its identification may fail when malware is set off by command line. For instance, it may not change to its dangerous execution state. Another downside of malware sandboxes is their short guard time; as a rule, sandbox may neglect to stand by adequately long enough, and they may not record all happenings. For instance, if the malware is set to stand by 10 min prior to it carrying out dangerous event, this time-set off activity may not be identified.

Prior to executing, some malwares assess the existence of specific registry keys or files on the system, which do not exist in the sandbox. The shortfall of such authentic information forestalls malware from running inside the sandbox environment. The sandbox environment OS might be deceitful in light of the fact that the malware is developed for a particular OS; that is, the malware may crash on Windows XP, while it might execute on Windows 7. A sandbox can report essential functionality, yet it cannot characterize the file as dangerous. The produced reports should be assessed to ascertain whether the file is dangerous.

Malwares can execute some functionality to identify VM platforms, and they can adjust their behaviour. One renowned task executing this identification functionality is the Red Pill venture of Joanna Rutkowska, which is the most known VM-aware program, explains the attainability and simplicity of such identification approaches (Rutkowska, 2019). In addition, a few tools (for instance Scoopy (Scoooyng, 2020) utilise VM-based opcodes or inquiry VM-explicit virtual registry values. Despite the fact that there is an enormous expansion in the quantity of VM-based malwares, not many measure of scholastic examination and work has been implemented to interrupt those malwares and examine them dynamically. As expressed in (Ferie, 2020), VMs are more dependable and not straightforward with respect to timing-based identification contrasted and emulator systems like QEMU. Hence, it is significant and crucial to have the total and coordinated devices to examine VM-based malwares.

### 2.6 Techniques malwares use to detect Virtual Machine environments.

Mechanisms

OS Features

Operating system acts as an intermediary between a user of a computer and the computer hardware Operating system goals. These goals range from executing user programmes and providing solutions to user problems to optimizing the efficiency of the hardware. The structure of the computer system suppoorts these goals. Some of the techniques used by malware are programmed to expoilt vulnerabilities in this structure. The computer system can be divided into three distinct groups: Operating System (OS), the application programmes and the users. The operating system oversees the allocation of resources amongst various applications and users while the applications use resources provided by the OS to solve the queries of the user which could be humans or other machines. Malware programmers can insert codes to collect information on OS.

Generic OS queries include;

1. Specifying username

3. Specifying the host name

4. Specifying whether the total RAM is low

5. Specifying the screen resolution is non-usual for host OS

6. Specifying the number of processors

Global OS Objects Detection

The principle of all the global objects detection methods is the following:

There are no such objects in usual host; however, they exist in particular virtual environments and sandboxes. Virtual environment may be detected if such an artifact is present. (Checkpoint, 2021)

1. Specifying global mutexes

2. Checking specific virtual devices

3. Checking specific global pipes

4. Specifying global objects

5. Checking object directory and virtual registry

Code Sample

|  |  |  |
| --- | --- | --- |
| // usage sample:  supIsObjectExists (L"\\Driver", L"SbieDrv"); // sample values from the table below  typedef struct \_OBJECT\_DIRECTORY\_INFORMATION {  UNICODE\_STRING Name;  UNICODE\_STRING TypeName;  } OBJECT\_DIRECTORY\_INFORMATION, \*POBJECT\_DIRECTORY\_INFORMATION;  BOOL supIsObjectExists (  \_In\_ LPWSTR RootDirectory,  \_In\_ LPWSTR ObjectName)  {  OBJSCANPARAM Param;  if (ObjectName == NULL) {  return FALSE;  } | PARAM. Buffer = ObjectName;  Param.BufferSize = (ULONG)\_strlen\_w(ObjectName);  return NT\_SUCCESS (supEnumSystemObjects (RootDirectory, NULL, supDetectObjectCallback, &Param));  }  NTSTATUS NTAPI supDetectObjectCallback (  \_In\_ POBJECT\_DIRECTORY\_INFORMATION Entry,  \_In\_ PVOID CallbackParam  )  {  POBJSCANPARAM Param = (POBJSCANPARAM)CallbackParam;  if (Entry == NULL) {  return | STATUS\_INVALID\_PARAMETER\_1;  }  if (CallbackParam == NULL) {  return STATUS\_INVALID\_PARAMETER\_2;  }  if (Param->Buffer == NULL || Param->BufferSize == 0) {  return STATUS\_MEMORY\_NOT\_ALLOCATED;  }  if (Entry->Name.Buffer) {  if (\_strcmpi\_w (Entry->Name.Buffer, Param->Buffer) == 0) {  return STATUS\_SUCCESS;  }  }  return STATUS\_UNSUCCESSFUL;  } |

GitHub. 2021. GitHub - hfiref0x/VMDE: Source from VMDE paper, adapted to 2015. [online] Available at: <https://github.com/hfiref0x/VMDE> [Accessed 10 Nov 2021].

UI Prototyping

User interface prototyping is a rapid analysis technique where users are actively involved in the mocking-up a system. UI prototypes serve:

As an examination antique that empowers you to investigate the issue space with your partners.

As necessities antique to at first imagine the framework.

As a plan antiquity that empowers you to investigate the arrangement space of your framework.

A vehicle for you to convey the conceivable UI design(s) of your framework.

A possible establishment from which to keep fostering the framework (assuming you expect to discard the model and begin once again without any preparation then you don\'t have to contribute the time composing quality code for your model).

Recognition techniques

1. Check whether windows with specific class names are available in the operating system

2. Check if high level windows\' number is missing

Static UI Artifacts

These are a class artifact relentless in nature, and consequently give a more elevated level of sureness while testing for the presence of guarded devices.

Registry

The windows library is a various leveled information base putting away Windows and other application settings. Practically totally introduced programming makes library keys and qualities.

Windows library contains a huge measure of data about numerous parts of a framework, including numerous that would fit in the previously mentioned classifications.

Ransomware, for example, Tesla Crypt and Locky are genuine instances of malware looking for AV items by this key.

WMI

Windows Management Instrumentation (WMI) is the foundation for the executive’s information and procedure on Windows-based working frameworks. You can compose WMI contents or applications to computerize regulatory assignments on distant PCs.

Normal Conventional WMI questions incorporate;

 Ascertain whether number of processors is low

 Check whether hard disk size is small

 Check addresses

 Check chip temperature

Standard COM functions sequence

1. Start COM initialization: Initialize/CoInitializeEx

2. Co-Create the required interface instance: Cocreate Instance/CoCreateInstanceEx

3. Connect to the particular services via the interface instance with the following function: Connect Server

4. Get methods for the services and set their arguments with these functions:

5. Get information from the services and execute the methods of the services with the functions below. The functions on the left are proxies for the functions on the right - which are called internally:

ExecQuery -> IWbemServices\_ExecQuery

ExecMethod -> IWbemServices\_ExecMethod

ExecMethodAsync -> IWbemServices\_ExecMethodAsync (GitHub, 2015).

File Systems

Practically any introduced program keeps in touch with itself to the plate. Already we saw that exploit units stop the contamination interaction within the sight of explicit documents – however numerous different sorts of malware end its execution too.

This for instance is the primary capacity of the IRONGATE SCADA focused on malware, note that before it performs vindictive action, it calls the capacity distinguish VMware (Homan, et al., 2017)

UI/Dynamic Artifacts

This is a class of artifact that contains temporary information. Generally, searching for these indicators is not as effective as static ones, but it enables detection of a wider range of products that don't have traceable files or registry artifacts.

Processes

"A process is an executing program", as Microsoft characterizes it. While investigating a malware, either physically or consequently, the investigator utilizes many projects – not concealed naturally. In addition, security items, for example, hostile to infection typically have various cycles zeroed in on ensuring the machine from one perspective, and showing a decent UI on the other.

Here’s Kaspersky's AV cycle model:

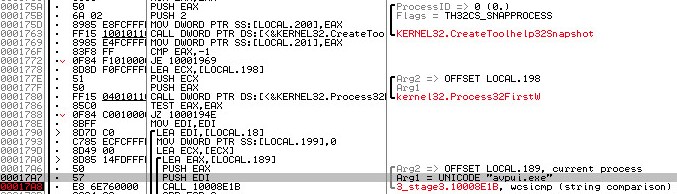
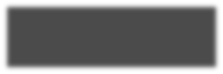


Figure 5: USB Thief checking avpui.exe process

Handles

Handles are a process entry structures.

Sample Malware accessing windows as shown

~ Mark Vincent Yason at Black Hat USA way back in 2007:[[1]](#footnote-1)

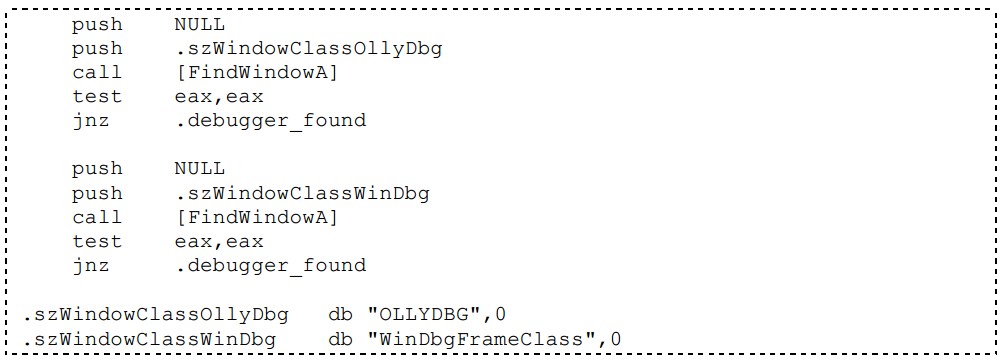
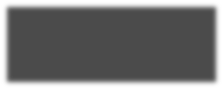


Figure 6:Detecting OllyDbg and WinDbg

FindWindowA API will return true if OllyDbg or WinDbg debuggers are on.

User Name

Some sandbox solutions and malware experts show little innovativeness while choosing a username for their investigation machine.

Malware exploits this absence of imagination and can undoubtedly test it against a boycott before unloading itself.

In the example bellow we see a malware call the GetUserNameA:

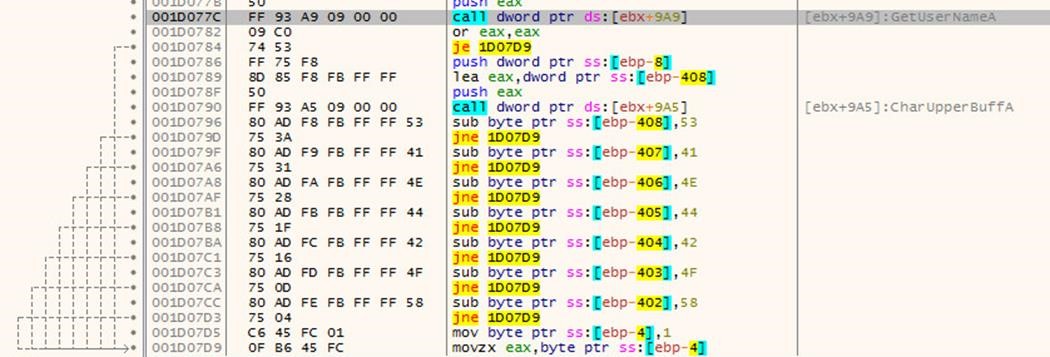
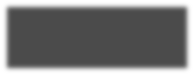


Figure 7:Detecting sandbox of a current user's

Human Like Behavior "Turing Tests"

If a malware is executed on a machine that has no man behind it, the mouse cursor will stay at the same position. This is countered by randomly moving the mouse around the screen, as is implemented in many sandboxes.

Hardware

Common evasion entry techniques and methods include;

1. Checking for HDD specific name

2. Checking if HDD Vendor ID has specific value

3. Checking presence of audio device

4. CPU temperature information

5. Checking physical display adapter interfaces

Joanna Rutkowski’s Original Red Pill

This technique was introduced by a Polish security researcher named Joanna Rutkowski, to detect virtualized environment. It depicts correlation between the location of the interrupt descriptor table (IDT) in memory, and virtualization products.

Every machine running a VM has only a single register holding the address of the IDT, but at least two OS (host and guest) sharing memory between them. The virtual machine has to prevent collisions between them, so it allocates the guest IDT to constant addresses. (Ou,, 2016)

CPU

There are numerous ways to detect whether the malware is running in a VM by using x86 instructions or low-level Windows API functions.

Examples are:

1. CPUID instruction distinct values
2. Time values for virtual machine calls

Time-based sandbox evasion techniques include;

* Delayed execution e.g., Use of Task Scheduler
* Sleep skipping i.e time intervals using different methods
* Getting the current date and time
* Difference in time
* Last boot time methods

Firmware

These are special memory areas used by operating systems which contain specific values if OS is run under virtual environment. These memory areas may be dumped using different methods depending on the OS version.

Firmware sample tabling:

typedef struct \_SYSTEM\_FIRMWARE\_TABLE\_INFORMATION {

ULONG Provider Signature;

SYSTEM\_FIRMWARE\_TABLE\_ACTION Action;

ULONG TableID;

ULONG TableBufferLength;

UCHAR Table Buffer [ANYSIZE\_ARRAY]; // <- the result will reside in this field

} SYSTEM\_FIRMWARE\_TABLE\_INFORMATION, \*PSYSTEM\_FIRMWARE\_TABLE\_INFORMATION;

// helper enum

typedef enum \_SYSTEM\_FIRMWARE\_TABLE\_ACTION

{

SystemFirmwareTable\_Enumerate,

SystemFirmwareTable\_Get

} SYSTEM\_FIRMWARE\_TABLE\_ACTION, \*PSYSTEM\_FIRMWARE\_TABLE\_ACTION;

(GitHub, 2021).

Firmware detection methods includes use of;

1. Specific strings present in Raw Firmware Table

2. Specific strings that are present in Raw SMBIOS Firmware Table

Network

Network avoidances can sidestep network interruption location/counteraction frameworks to convey exploits, assaults, or malware to casualties without being recognized.

Language structure for change of TCP streams to codeword streams is proposed to work with the extraction of factual highlights while protecting the avoidance conduct ascribes of unique organization streams.

A TCP stream is then changed to a decent length numeric element vector. Administered multi-class classifiers are based on the extricated include vectors to separate various sorts of avoidances from typical streams.

Quantitative assessments on an avoidance dataset comprising of ordinary organization streams and eight sorts of nuclear avoidance streams showed that the proposed approach accomplished a reassuring presentation with a precision of 98.95% (Jingping, Chen Kehua, Chen Jia, Zhou Dengwen, June 2019)

Network detection methods used include;

1. Using network properties such as MAC address, Adapter name

2. Checking network security perimeter

3. NetValidateName result based anti-emulation technique

4. Use of Cuckoo Result Server connection based anti-emulation technique

Code sample (function GetAdaptersAddresses)

|  |  |
| --- | --- |
| GetAdaptersAddresses (AF\_UNSPEC, ...)  GetAdaptersInfo  Code sample (function GetAdaptersAddresses)  int pafish\_check\_mac\_vendor (char \* mac\_vendor) {  unsigned long alist\_size = 0, ret;  ret = GetAdaptersAddresses (AF\_UNSPEC, 0, 0, 0, &alist\_size);  if (ret == ERROR\_BUFFER\_OVERFLOW) {  IP\_ADAPTER\_ADDRESSES\* palist = (IP\_ADAPTER\_ADDRESSES\*) LocalAlloc (LMEM\_ZEROINIT, alist\_size);  void \* palist\_free = palist;  if (palist) {  GetAdaptersAddresses(AF\_UNSPEC, 0, 0, palist, &alist\_size); | char mac [6] ={0};  while (palist){  if (palist->PhysicalAddressLength == 0x6) {  memcpy (mac, palist->PhysicalAddress, 0x6);  if (! memcmp(mac\_vendor, mac, 3)) { /\* First 3 bytes are the same \*/  LocalFree(palist\_free);  return TRUE;  }  }  palist = palist->Next;  }  LocalFree(palist\_free);  }  }  return FALSE;  } |

GitHub. 2016. GitHub - a0rtega/pafish: Pafish is a testing tool that uses different techniques to detect virtual machines and malware analysis environments in the same way that malware families do. [online] Available at: <https://github.com/a0rtega/pafish> [Accessed 18 November 2021].

Macos Sandbox Detection Methods

Most macOS-specific methods for sandbox detection are based on using shell commands such as “sysctl” and “ioreg”

Example of hardware model detection method -sysctl -n hw. model

Example using, I/o kit registry to detect virtual os - ioreg -rd1 -c IOPlatformExpertDevice

Sample output on native Apple hardware:

$ ioreg -rd1 -c IOUSBHostDevice | grep "USB Vendor Name"

"USB Vendor Name" = "Apple Inc."

"USB Vendor Name" = "Apple Inc."

"USB Vendor Name" = "Apple, Inc." (CheckpointResearches, 2021)

Anti-Evasion Techniques

Concealment

Sandboxes seek to mirror the execution of an example in a genuine machine, and on the off chance that it effectively distinguishes a sandbox environment, the sandbox will have apparently killed its main goal. Anti-virus solutions may classify some evasive techniques as suspicious, enabling them to detect malware by usage of evasive techniques.

Prevention

Malware almost by definition is paranoid, it is trying to avoid detection by employing many advanced evasion techniques which constitute its core strength.

The concept of adding artifacts sought by malware is somewhat similar to the classic "vaccination" idea (Minerva Labs' ,1990)

Evasive techniques are just one aspect of malware. At the moment we are at crossroads in direct arms race between "good and evil", with each new malware coming up with sophisticated tests prior to the deployment of a payload. Internal competition between "malware vendors" just increases the numbers of techniques added to malware, as "clientele" often prefer low-end products.

This explosion in the number of evasive techniques looks frightening at first sight but it creates new opportunities for defenders as well.

Every researcher should now delve into the question” Does attacker stop while trying to evade detection or perhaps give up and get caught”.

2.6 Summary

This chapter has looked at the types of malware and malware detection techniques. The strengths and limitations of the detection techniques were looked at. In the present study, virtual machines with a slight difference in their configurations will be set up in order to bypass the malware’s detection techniques. The data to be obtained will be in qualitative form. The data set will record each virtual machine, if a particular malware ran on each of the virtual machine, the path or directory in which the malware stores itself, if the saved passwords on the virtual machines are seen by each malware. The steps/procedure are detailed in the next chapter.

# CHAPTER THREE METHODOLOGY

## 3.1 Introduction

These days, digital world is being improved by an enormous assortment of computerized data innovation based services. An expanding degree of virtual and mobile tools utilisation prompts an amazing reliance on data security. Investigation and identification of malicious programs or otherwise called malware is a challenging task because of the presentation of innovative obfuscation methods by malware creators. In this investigation, we mostly focus on virtual machine methods to give secure and reproducible environments for malware examination and its execution problems.

This chapter will detail the procedure to be used in finding out how malwares detect virtual machines. As mentioned earlier, VMs are good environments where malware can be analysed, however most modern malware can detect the environment in which they are running and refrain from executing actions meant for a victim’s device. In a bid to achieve the aims of this research which is to investigate the prevalence of virtual machine evasion techniques in malware. The paper will draw on results of relevant research listed so far in the previous sections also the paper will try to determine presence of these techniques in more modern malware samples by implementing an experimental design. Before diving into the relevance of the method chosen, this paper will consider a more sensitive topic: Ethical considerations.

## 3.2 ETHICS CONSIDERATION

Ethics in malware researches has long been a debate among malware researches. Some of these ethical problems or dilemmas as listed by John includes how human subjects are treated and malware and information ethics. For this experiment, no human subject was used. Malware and information ethics face several dilemmas as researchers try to form generalised ethics norms that are applicable to information technology. One of the dilemmas is how information is communicated. Malware researches conflicts with the high values academics has placed on the free sharing of research findings. This is dilemma is caused by the sensitive nature of malware researches. For example, publishing finding on an exploit might prove dangerous. As malware researchers, it's tricky to maintain this balance. It is a good suggestion in the case of this example to first alert the software owner whose product has been exploited and give them some time so they can fix a patch for the exploit before publishing your findings. Appropriate permissions were sought before choosing and undertaking this project for an IT security Master thesis. The permission was sought through my school assigned research supervisor. The research was conducted in a controlled environment, isolated form the network to avoid accidental spread of virus. The machines were restored after each malware execution to avoid interference from unseen ruminants of the previous malware. The materials used were selected from commercially available sources which makes the experiment relatable to real life scenarios. The research was careful not to publish any results or links that will be deemed dangerous in the hands of an attacker.

## 3.3 PROCEDURES AND PRECAUTION

## 3.3.1 MATERIALS

For this experiment, the following materials were used.

Machines: Windows 7 OS on Virtual Private server, Windows 10 OS on Virtual Private server, VirtualBox

One PC was used to remotely connect to VMs hosted on virtual private services AND Windows 7 and 10 OS hosted on Virtual Private Servers.

Virtual Machines: Two VPS services were bought online. They both had admin rights activated by the vendor. This allows for easy manipulation of the machines. VirtualBox was also installed with the operating systems mentioned.

One PC was used to remotely connect to VMs hosted on virtual private services. The specification of the PC included a hard drive size of 256GB, 16 GB RAM running on MacOS. Two VPS services were sourced online. The first had Windows 7 professional installed on it while the second had windows 10 installed on it. This was done to observe the relationship between the OS and the malware behavior. Also, the research hoped it will provide insight into the difference between old and contemporary malware practices. They both had admin rights activated by the vendor. This allowed for easy manipulation of the machines. Virtualbox was also installed with the operating systems mentioned. Malwares samples: Samples of malwares were obtained from virusshare.com.

After the setup of the VM with their normal configuration, the MAC address was noted and confirmed to be between the range of 00-03-FF-00-00-00 to 00-03-FF-FF-FF-FF. This is the typical range for MAC addresses used by VMs. Then a restore point was created. After the creation of a restore point, the malware was then executed one at a time.

## 3.4.2 STEPS TAKEN

Upon execution of malware, its activities are recorded. An activity in this context is either when a process tree is started, the files stay with the task manager for a specific window of time or other evidence of infection. When an incident is noticed for any activity, it is represented by 1. When there are no activities which are judged when the malware stays a short while in the task manager and self-deletes, it is valued as 0. The computer was restored to the created restore point before each execution. Twenty malware samples were executed in VirtualBox containing windows 7 and 10 OS and then on windows 7 and 10 OS hosted on a private server.

# 

# CHAPTER FOUR

# 4.1 RESULTS AND ANALYSIS

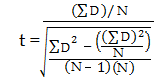
The research found that most of the malware examined exhibited one or more of the activities stated earlier when executed on Windows 7 and 10 environments when compared to the behaviors on each of the different OS on VirtualBox. The study conducted this test on twenty different samples it identified from the malware samples obtained. (See Appendix 1 for full table). About 15% of the samples showed activities of anti-VM behavior. While about 80% showed no significant sign of anti-VM activity. One of the malware samples was unresponsive in all environments.

In Fig 4.1 below, the difference in the test on window 7 OS (CONFIG 1) and windows 7 OS on VirtualBox (VIRTUALBOX) shows minor difference in the behaviors between the two environments with 16 samples executing and/or creating processes while 3 showed evidence of self-destruction thereby exhibiting anti-VM properties. Subsequent results from the test on windows 10 VPS and windows 10 on VirtualBox yielded the same results as in Fig 4.1.



Fig 4.1. Chart showing test conducted on windows 7 environment.

Although it shows the existence of virtual machine detection techniques it is not substantial enough to be called widespread for the modern malwares tested from the sample set selected. To determine the significant difference in the mean for results of the windows 7 environment, a Paired T test formula was used as shown



In this study the t= 7.5 which indicates a stark difference rate. However, the consistency in the results in both windows 7 and 10 operating systems highlights the irrelevance of the operating system in the anti-virtual machine methods.

## 4.2 LIMITATIONS TO STUDY

Time was the major limiting factor in this study. Also, the sheer number of malwares available today makes it a herculean task to finish the study within the given time. This makes the data too small to generalise. To mitigate this, the researcher ensured to use popular malware with high infection rate. This will help make the experiment a bit more applicable in real world scenarios. Also, the study will discuss other similar works, experiments and results in the subject area of this research to reach a substantial conclusion.

## 4.2 STUDY DISCUSSION

However small the size sample for the experiment, evidence of anti-VM techniques can be noted in some of the samples tested. This made no difference if it was a newer version OS (windows 10) or an older OS (windows 7). Although the experiment was not able to prove that malware behaved differently as a result of the OS, the study might suggest that malware do not use OS based commands for its techniques. Several researchers argue that a lot of malwares utilise one of these methods in their process. One of these includes study by Chen which tested over 17000 malware specimens and compared the presence of anti-debugging and anti-malware use by the malware in a similar experiment (Chen et al., 2016). From the result of Chen’s study which was conducted with generic and advance persistent threat (APT) level malwares, it can be noted that majority of malwares exhibited anti-debugging or anti-virtual malware characteristics. Branco, whose study is more recent created a taxonomy for malwares that exhibit anti-VM traits. That study concludes that anti-VM techniques was found in over 80% of the over 4 million samples of malwares they tested. This is the most significant study that this research found for this analysis. Despite these proofs, other researches (Mushtaq, 2011) argue that anti VM techniques is not widespread because it can be reversed and used to identify malwares. In his research Mustaq concluded that out of top twenty malware that are mostly used by attackers, only one exhibited evasive or anti-VM traits. The study further argued that anti-VM methods may expose the malware to easier detection by a malware analyst. For example, if a malware has a string that looks for the virtual box core files, that code can be used to identify that element as being evasive.

# CONCLUSION

This study observed how different malware behaved in virtual machines in a bid to understand how malwares detect virtual machines. The study consisted of setting up virtual machines with a slight difference in their configurations aimed to bypass the malware’s detection techniques. The study was directed by four specific objectives which included to: determine which malware use virtual machine (VM) detection techniques, determine the techniques these malwares use in detecting virtual machines, use obfuscating methods to configure virtual machines in which malwares will run as they would on a real machine, find out the usefulness of obfuscation as a countermeasure technique. The study was further guided by three formulated research questions, which addressed as: What other techniques do malwares employ to detect virtual machines?; Which malwares exhibit this behaviour? And What is the efficacy of obfuscation as a countermeasure to virtual machine detection by malwares?

Three out of the twenty malwares tested exhibited virtual machine detection and one was unresponsive to the test. Although it showed the existence of virtual machine detection techniques it was not substantial enough to be called widespread for the modern malwares tested from the sample set selected. However, in a surprising turn of events, when the same experiment was conducted on windows 10 environment it produced the same results. As expected, the study had some constraints, of which time was the major constraint in the study. The results and findings from this study is similar to those of Ping et al. (2016) and Mushtaq (2011). The study would benefit from an automated system for detection of anti-VM techniques in malware samples. This would help accommodate a larger sample size for analysis in less time. Further research on this topic could also focus on the prevalence of a particular anti-WM technique.

# REFERENCES

Böhne L. Pandora’s Bochs (2008). Automatic unpacking of malware. M.Sc. Thesis, Dept. Computer. Science, Rwthaachen Univ., Aachen, Germany

Checkpoint, 2022. Evasions: Global OS Objects. [online] Evasion techniques. Available at: <https://evasions.checkpoint.com/techniques/global-os-objects.html> [Accessed 10 January 2022].

2006, T., Liston, Skoudis and Liston, T. (2006). *On the Cutting Edge: Thwarting Virtual Machine Detection*. [online] . Available at: https://handlers.sans.org/tliston/ThwartingVMDetection\_Liston\_Skoudis.pdf.

Anubis (n.d.). *iSecLab - Welcome*. [online] www.iseclab.org. Available at: http://www.iseclab.org [Accessed 13 Aug. 2021].

Bayer, U. (2005). *TTAnalyze: A Tool for Analyzing Malware*. [online] . Available at: https://repositum.tuwien.at/bitstream/20.500.12708/12822/2/Bayer%20Ulrich%20-%202005%20-%20TTAnalyze%20a%20tool%20for%20analyzing%20malware.pdf [Accessed 13 Aug. 2021].

Ben Flanagan. Former CIA chief speaks out on Iran Stuxnet attack. 2011. url: http://www.thenational.ae/business/industry-insights/technology/former-cia-chief-speaks-out-on-iran-stuxnet-attack.

Matrosov Aleksandr, Rodionov Eugene, Harley David, and Malcho Juraj. Stuxnet Under the Microscope. 2011. url: http://www.eset.com/us/resources/white-papers/Stuxnet\_Under\_the\_Microscope.pdf.

Charette, S. (2016). *Programming Comments - How not to detect virtualization*. [online] No-ip.com. Available at: http://charette.no-ip.com:81/programming/2009-12-30\_Virtualization/ [Accessed 13 Aug. 2021].

*Cisco.( 2014). Annual Security Report*. url: http://www.cisco.com/web/offer/ gist\_ty2\_asset/Cisco\_2014\_ASR.pdf.

CuckooSandbox (2020). *Cuckoo Sandbox - Automated Malware Analysis*. [online] cuckoosandbox.org. Available at: <https://cuckoosandbox.org>.

GitHub. 2015. GitHub - LordNoteworthy/al-khaser: Public malware techniques used in the wild: Virtual Machine, Emulation, Debuggers, Sandbox detection.. [online] Available at: <https://github.com/LordNoteworthy/al-khaser> [Accessed 11 November 2021].

Evasion techniques. 2017. macOS sandbox detection methods. [online] Available at: <https://evasions.checkpoint.com/techniques/macos.html> [Accessed 1 December 2021].

Ferrie, P. (2007). *Attacks on Virtual Machine Emulators*. [online] www.semanticscholar.org. Available at: https://www.semanticscholar.org/paper/Attacks-on-Virtual-Machine-Emulators-Ferrie/56c478351657e0ec0106520c8cca82f93991ad61 [Accessed 13 Aug. 2021].

Homan. J, McBride. S, Caldwell. R (2016) IRONGATE ICS Malware: Nothing to See Here...Masking Malicious Activity on SCADA Systems. Available at: https://www.fireeye.com/blog/threat-research/2016/06/irongate\_ics\_malware.html [Accessed 10 September 2021].

Ilsun You and Kangbin Yim. “Malware Obfuscation Techniques: A Brief Survey.” In: *BWCCA*. 2010, pp. 297–300.

Jeong, G., Choo, E., Lee, J., Bat-Erdene, M. and Lee, H. (2010). *Generic unpacking using entropy analysis*. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/5665789 [Accessed 13 Aug. 2021].

Jovanovic, B., 2021. A Not-So-Common Cold: Malware Statistics in 2021 | DataProt. [online] DataProt. Available at: <https://dataprot.net/statistics/malware-statistics/> [Accessed 12 October 2021].

Kang, M.G., Poosankam, P. and Yin, H. (2007). Renovo. *Proceedings of the 2007 ACM workshop on Recurring malcode - WORM ’07*.

Kusnetzky, D., 2011. *Virtualization*. Sebastopol, CA: O'Reilly.

Barriga J.J. and Yoo S.G. 2017, ‘Malware Detection and Evasion with Machine Learning Techniques: A Survey’, *International Journal of Applied Engineering Research.* Vol 12, Number 18 (2017) pp. 7207-7214

Ligh, M., Adair, S., Hartstein, B. and Richard, M. (2010). *Malware Analyst’s Cookbook and DVD: Tools and Techniques for Fighting Malicious Code*. [online] *Google Books*. John Wiley & Sons. Available at: https://www.google.co.uk/books/edition/Malware\_Analyst\_s\_Cookbook\_and\_DVD/HbRTDwAAQBAJ?hl=en&gbpv=1&dq=Malware+Analyst [Accessed 13 Aug. 2021].

Moser, A., Kruegel, C. and Kirda, E. (2007). *Limits of Static Analysis for Malware Detection*. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/document/4413008.

Norman (2021). *Norman | Antivirus & Security Software for Home & Business*. [online] AVG.com. Available at: https://www.norman.com/en-ww/homepage [Accessed 13 Aug. 2021].

Oracle (2019). *Oracle VM VirtualBox Overview*. [online] Oracle.com. Available at: <https://www.oracle.com/assets/oracle-vm-virtualbox-overview-2981353.pdf>.

Ou, G., 2016. *Blue Pill: The first effective Hypervisor Rootkit | ZDNet*. [online] ZDNet. Available at: <https://www.zdnet.com/article/blue-pill-the-first-effective-hypervisor-rootkit/> [Accessed 10 November 2020].

Philip O’Kane, Sakir Sezer, and Kieran McLaughlin. “Obfuscation: The hidden malware”. In: *IEEE Security & Privacy*9.5 (2011), pp. 41–47.

QEMU (2020). *QEMU*. [online] wiki.qemu.org. Available at: http://wiki.qemu.org [Accessed 13 Aug. 2021].

Quist, D. and Smith, V. (n.d.). *Detecting the Presence of Virtual Machines Using the Local Data Table*. [online] . Available at: https://www.ccoderun.ca/programming/2009-12-30\_Virtualization/www.offensivecomputing.net\_vm.pdf [Accessed 13 Aug. 2021].

Reeds, J. (2019). *What Is Hyper-V Manager and How to Use It: A Complete Guide*. [online] www.nakivo.com. Available at: https://www.nakivo.com/hyper-v-backup/what-is-hyper-v-manager-and-how-does-it-work/ [Accessed 13 Aug. 2021].

Regshot (2020). *Google Code Archive - Long-term storage for Google Code Project Hosting.* [online] code.google.com. Available at: https://code.google.com/archive/p/regshot/ [Accessed 13 Aug. 2021].

Rid, T. and McBurney, P. (2012). Cyber-Weapons. *The RUSI Journal*, 157(1), pp.6–13.

Rosencrance, L. (2014). *What is sandbox (computer security)? - Definition from WhatIs.com*. [online] SearchSecurity. Available at: https://searchsecurity.techtarget.com/definition/sandbox.

Schipka, M. (2009). Dollars for downloading. *Network Security*, [online] 2009(1), pp.7–11. Available at: https://www.sciencedirect.com/science/article/pii/S1353485809700066 [Accessed 13 Aug. 2021].

scoobyng (2019). *ScoopyNG — The VMware detection tool*. [online] www.trapkit.de. Available at: https://www.trapkit.de/tools/scoopyng/ [Accessed 13 Aug. 2021].

Sharmai, A. and Sahay, S. (2014). *(PDF) Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey*. [online] ResearchGate. Available at: https://www.researchgate.net/publication/263507074\_Evolution\_and\_Detection\_of\_Polymorphic\_and\_Metamorphic\_Malwares\_A\_Survey.

Skoudis, E. and Zeltser, L. (2008). *Malware fighting malicious code*. Upper Saddle River, Nj Prentice Hall Ptr.

Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P. and Saxena, P. (2008). BitBlaze: A New Approach to Computer Security via Binary Analysis. *Information Systems Security*, pp.1–25.

stevewhims (n.d.). *WMI Classes - Win32 apps*. [online] docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-classes [Accessed 13 Aug. 2021].

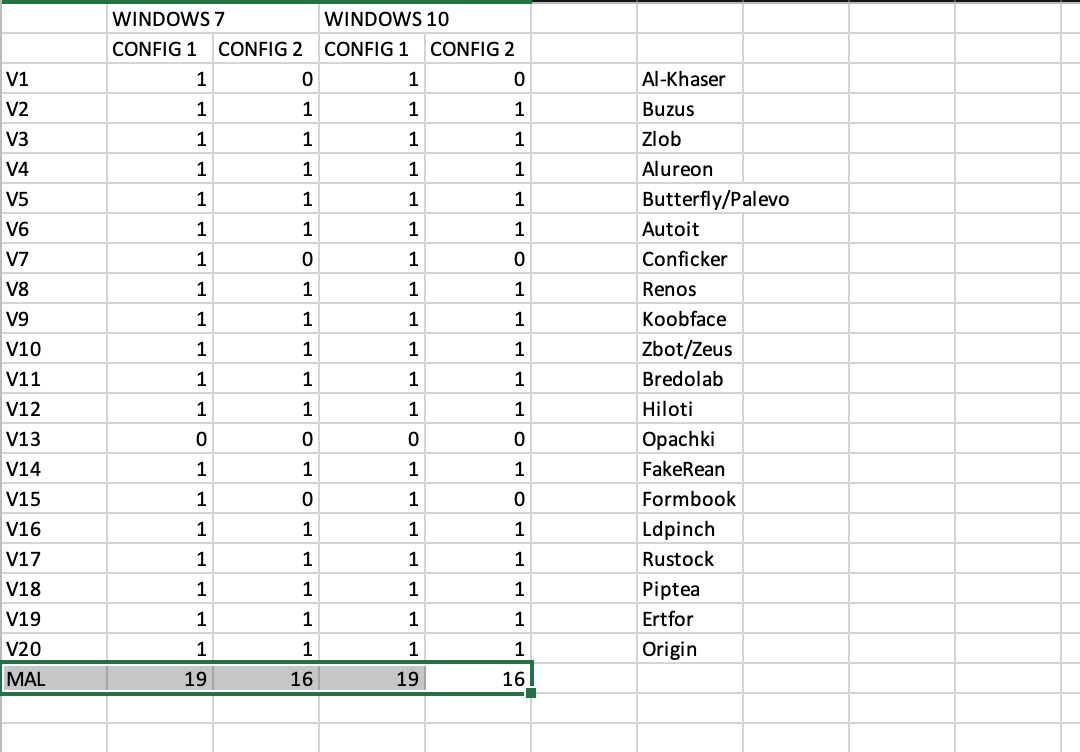
Willems, C., Holz, T. and Freiling, F. (2007). Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy Magazine*, 5(2), pp.32–39.

www.cexx.org. (2016). *Adware, Spyware and Advertising Trojans - Info & Removal Procedures*. [online] Available at: http://www.cexx.org/adware.htm.

Yin, H. and Liang, Z. (2019). *HookFinder: Identifying and Understanding Malware Hooking Behaviors*. [online] . Available at: http://bitblaze.cs.berkeley.edu/papers/hookfinder\_ndss08.pdf [Accessed 13 Aug. 2021].

‌

# APPENDICES

APPENDIX 1. Table of results

1. [↑](#footnote-ref-1)