

**BEHAVIOR BASED MALICIOUS SOFTWARE DETECTION AND
CLASSIFICATION**

**(DAVRANIŞ TABANLI ZARARLI YAZILIM TESPİTİ VE
SINIFLANDIRILMASI)**

by

Abdurrahman PEKTAŞ, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

Date of Submission : May 25, 2012

Date of Defense Examination : June 22, 2012

Supervisor : Assoc. Prof. Dr. Tankut Acarman

Committee Members : Asst. Prof. Dr. Murat Akın

Dr. Hayretdin Bahşı

Acknowledgements

I would like to thank particularly to Assoc. Prof. Dr. Tankut Acarman for giving me the opportunity to work such an interesting project and also for giving interesting approaches that help me a lot. I am also grateful for his understanding of my time limitations and encourage he has provided.

I would like to present my deepest appreciation to all my colleagues at Biliřim ve Bilgi Gvenlięi İleri Teknolojiler Arařtırma Merkezi (BILGEM) for their great support and understanding. Finally, I want to thank my dear parents and my sisters for their love and support.

Abdurrahman PEKTAŐ

İstanbul, May 25th, 2012

Table of Contents

Acknowledgements	ii
Table of Contents	iii
List of Symbols	vi
List of Figures.....	viii
List of Tables.....	ix
Abstract.....	x
Résumé.....	xiii
Özet	xvi
1 Introduction.....	1
1.1 Thesis Organization	2
2 State of the Art	4
2.1 Types of Malwares.....	4
2.1.1 Backdoor.....	4
2.1.2 Trojan Horses	5
2.1.3 Rootkits.....	5
2.1.4 Viruses	6
2.1.5 Botnet.....	6
2.2 Malware Analysis Methods.....	7
2.2.1 Static Analysis	7
2.2.2 Dynamic Analysis	8

3	Anti-Virtual Machine Techniques	10
3.1	Hardware Fingerprinting	10
3.2	Registry Check	11
3.3	Memory Check.....	13
3.4	VMware Communication Channel Check	14
3.5	File & Process Check.....	16
4	Implementation of Dynamic Malware Analyzer	18
4.1	Binary Instrumentation Tool: Pin	18
4.1.1	Intended use of Pin.....	19
4.2	Monitor System.....	22
4.2.1	Monitor Processes	22
4.2.2	Monitor Connections.....	24
4.2.3	Monitor Services	25
4.2.4	Monitor Registry	26
4.3	Basic Configuration	28
4.3.1	Restrict Connectable Process	28
4.3.2	Restrict Destination Domain	29
4.4	Comparison of Dynamic Analysis Techniques.....	31
4.4.1	Overviewing Existing Dynamic Analysis Techniques	31
4.4.2	Pros & Cons of Our Dynamic Analysis Techniques.....	33
5	Implementation of n-gram Based Malware Classifier	35
5.1	Definition of n-gram	35
5.2	Existing n-gram Approaches to Analyze Malware	36
5.3	Our n-gram Based Malware Classifier	38
5.4	Obtained Results	42
6	Fusion	46

6.1	Evaluation Methodology & Obtained Results	48
7	Conclusion.....	52
	References.....	54
	Biographical Sketch	59

List of Symbols

DMA	: Dynamic Malware Analyzer
ASCII	: American Standard Code for Information Interchange
I/O	: Input/Output
MAC	: Media Access Control
OS	: Operating System
BIOS	: Basic Input-Output System
WMI	: Windows Management Instrumentation
SIDT	: Store Interrupt Descriptor Table
SLDT	: Store Local Descriptor Table
SGDT	: Store Global Descriptor Table
STR	: Store Task Register
IDT	: Interrupt Descriptor Table
API	: Application Programming Interface,
MSE	: Microsoft Security Essential
TR-CERT	: Turkey Computer Emergency Response Team
IRC	: Internet Relay Chat

IP : Internet Protocol

DDoS : Distributed Denial of Service

HTTP : Hyper Text Transfer Protocol

SMTP : Simple Mail Transfer Protocol

List of Figures

Figure 1.1. Overview of the Thesis Organization	3
Figure 3.1. Startup Programs Obtained from Registry	12
Figure 3.2. Snap Code of Red Pill Technique	14
Figure 3.3. VMware I/O Backdoor's Main Functionalities	15
Figure 3.4. Assembly Code to Detect VMware Machine via VMware I/O Port	15
Figure 3.5. Process Listing on a VMware Image	16
Figure 3.6. Search VMware Processes in Process List	17
Figure 4.1. Our Approach for Detecting anti-VM Aware Malware	20
Figure 4.2. Process Monitoring Feature of DMA	23
Figure 4.3. Network Monitoring Feature of DMA	25
Figure 4.4. Service Monitoring Feature of DMA	26
Figure 4.5. An Example of Registry Changes Made by Malware to Gain Persistence ..	27
Figure 4.6. A Malicious File Establishes Connection without User's Intent	29
Figure 4.7 DNS Resolver Script for Malware Domains	30
Figure 4.8. A Malicious Process Tries to Connect Malicious Domain	30
Figure 5.1. N-gram Sequences	35
Figure 5.2. Architecture of the Malware Classification System	40
Figure 6.1. Fusion of DMA and n-gram Based Classification Module	47
Figure 6.2. Details of Executed Malwares	49
Figure 6.3. Malware Detection Ratio	50
Figure 6.4. Number of Samples Detected by DMA's Modules	51

List of Tables

Table 3.1. Hardware Fingerprinting of Native and VMware Machine, [12]	11
Table 5.1 Number of the Instances for Each Subfamily	43
Table 5.2. Training Error	44
Table 5.3 Testing Error	45

Abstract

In recent years, cyber-attack, one of the most apparent subjects in social media, will continue to be on the agenda with increasing its importance. Malicious software which is used heavily in cyber-attacks has become indispensable object of the cyber war. Malicious softwares are used for different purposes such as steal sensitive information, create a backdoor to access system persistently, create a botnet in order to drive distributed denial of service attacks, etc. They may include one or more objectives and the objectives depend on the purpose of the malicious software writer. Stuxnet which contains three 0-day exploits and target to Iran's nuclear facilities, is a good example to show us that how malicious software may be used during cyber war.

As is known to all, to analyze malicious software first of all malware analyst need to detect these files and then classify them appropriately. The analysis methods are collected under two headings. These are static and dynamic analysis methods. In this study the new cascade malicious software detection method was developed by combining the mentioned two methods. In the study the dynamic analysis methods were used in order to detect malicious software. By the way static properties of the file were used during the classification phase.

Static analysis method is performed via only the static properties of the file and does not include the execution of the malicious file. In this study, we propose the new classification algorithm that uses byte sequences (n-gram) of the malicious file.

In dynamic analysis the malicious file is need to be executed in secure environment because it investigates the behavior of the malicious software. In dynamic analysis network connections, file system operations, the active processes on the system, etc. are tracked. To carry out the dynamic analysis we created a tool called as Dynamic Malware Analysis (DMA). The developed tool which is quite simple to use, can be run as system tray icon on the Windows operating system and it is capable of alerting the user if there is any malicious activity on the system.

Nowadays, malicious softwares use anti-debugger and anti-virtualization technique in order to prevent detection by dynamic analysis methods. In the study the dynamic analysis was performed on VMware virtualization environment. To bypass the anti-virtualization methods used by malwares the Pin tool was used. It is a free tool provided by Intel for the dynamic instrumentation of programs. With the help of this tool the processes, files and registry keys searching methods which are used by malicious software can be bypass.

The following features are considered when analyzing malicious software:

- Network connection changes on the system
- Registry changes
- Process changes
- Service changes

After detection step, the found malicious softwares are classified by using n-gram based malware classification mechanism. In this method, each malicious file is pointed by n-gram vector (byte sequences obtained from malicious file) of the file and the classification is performed over these vectors.

To evaluate the developed malicious software detection and classification modules the public Pahadus malware set was used and promising results were obtained. We obtained 86% accuracy over the specified public malware set. After the detection of the file n-gram based classification method was used. In the learning phase of the classification algorithm the malicious softwares which are provided by BILGEM were used. We were obtained 92% success rate when we choose n as 4 and L as 60.

Résumé

Ces dernières années les médias sociaux sont sujets à de nombreuses cyber-attaques et leurs nombres semblent augmenter chaque année et dans tous les secteurs. Les logiciels infectés sont les plus utilisés pour les attaques et sont devenus l'outil indispensable dans la guerre cybernétique. Ces logiciels sont utilisés pour le vol de données personnelles, créer des «backdoor» pour les accès aux systèmes persistant, créer des «botnet» pour mettre hors service les machines par déni de service. Les attaques sont menées pour différentes raisons et dépendent de l'intention du créateur du logiciel. Stuxnet exploite une vulnérabilité 0-day et avait prit pour cible les exploitations nucléaires de l'Iran, c'est un exemple type pour nous montrer les utilisations de logiciels infectés durant une guerre cybernétique

Comme nous le savons pour l'analyse d'un logiciel infecté il faut tout d'abord le détecter. Ainsi, une fois que l'expert a détecté les fichiers, il doit les classifier. Les méthodes d'analyse sont de 2 catégories: les analyses statiques et dynamiques. La combinaison de ces deux méthodes mise en cascade a permis de trouver une nouvelle méthode pour la détection des logiciels malveillants. Les recherches représentent l'analyse dynamique et la classification des logiciels infectés représente l'analyse statique.

La méthode d'analyse statique est exécutée seulement via les propriétés statiques du dossier et n'inclut pas l'exécution du dossier malveillant. Dans cette étude, nous

proposons un nouvel algorithme de classification qui utilise des séquences d'octet (n-gram) du dossier malveillant.

L'analyse dynamique doit se faire dans un environnement sécurisé puis le dossier doit être lancé pour analyser les comportements du logiciel malveillant. Toutes les analyses dynamiques sont suivies: les connexions réseaux, les opérations sur les fichiers systèmes, les processus actifs du système. Pour effectuer l'analyse dynamique nous avons créé un outil nommé «Dynamic Malware Analysis (DMA)». L'outil développé est simple d'utilisation et peut être exécuté dans le système d'exploitation Windows qui est capable de nous alerter s'il y a une activité malveillante en cours de fonctionnement.

De nos jours, les logiciels malveillants sont plus développés et utilisent des anti-debuggers et des techniques d'anti-machine virtuelle pour contrer la détection via l'analyse dynamique. Dans notre étude l'analyse dynamique est effectuée sous un environnement virtuelle VMware. Pour contrer les protections du logiciel malveillant l'outil Pin a été utilisé. C'est un outil libre d'utilisation mis en place par Intel pour les programmes d'instrumentation dynamique. Ainsi, à l'aide de cet outil la méthode de recherche du logiciel malveillant sur les processus, les dossiers et les registres est parée.

Les caractéristiques trouvées durant l'analyse du logiciel malveillant:

- Changement des connexions réseaux du système
- Changement des registres
- Changement des processus
- Changement des services

Après la détection, les logiciels infectés sont classés en utilisant n-gram pour définir la famille. Dans cette méthode chaque dossier infecté est ponté par un vecteur n-gram (la séquence est obtenu par le dossier infecté) et la classification est effectué via ce vecteur.

Pour évaluer le logiciel de détection et classification des modules nous avons utilisé le jeu de logiciel malveillant public Pahadus qui est en libre utilisation et nous avons obtenu des résultats très encourageant. Nous avons obtenu 86% d'exactitude sur la détection. Après la détection des fichiers nous avons utilisé la méthode de classification n-gram. L'algorithme sur la classification des logiciels malveillant nous a été fournit par BILGEM et nous avons obtenu 92% d'exactitude avec $n=4$ et $L=60$.

Özet

Son yıllarda sosyal medyada fazlaca yer alan konulardan biri olan “siber saldırılar” görünüşe bakılacak olursa önemini arttırarak gündemde kalmaya devam edecek. Siber saldırılarda etkin şekilde kullanılan zararlı yazılımlar siber savaşın vazgeçilmezi haline gelmiştir. Zararlı yazılımlar hassas veri çalma, sistemlere bağlantı için arka kapı oluşturma, servis dışı bırakma saldırılarında kullanılan (ddos) botnetler oluşturma gibi farklı amaçların birine ya da hepsine birden hizmet edebilmekte ve saldırganlar için gerekli ortamı sağlamaktadır. İran nükleer tesislerine düzenlenen ve daha önce tespit edilmemiş (0-day) 3 tane açıklığı barındıran stuxnet zararlı yazılımı siber saldırı dünyasında zararlı yazılımların ne derece etkin kullanıldığını göstermesi açısından güzel bir örnektir.

Bilindiği gibi zararlı yazılımların incelenebilmesi için öncelikle tespit edilmesi ve sınıflandırılması gerekmektedir. İnceleme yöntemleri iki başlık altında toplanır. Bunlar statik ve dinamik analiz yöntemleridir. Bu çalışmada bahsedilen iki yöntem birleştirilerek kademeli yapıda yeni bir zararlı yazılım tespit yöntemi geliştirilmiştir. Çalışmada dinamik analiz yöntemleri kullanılarak zararlı yazılımların tespit edilmesi sağlanmıştır. Zararlı yazılımların sınıflandırılması esnasında zararlı dosyanın statik özellikleri kullanılmıştır.

Statik analiz yöntemi zararlı dosyanın çalıştırılmadan sadece statik özellikleri göz önünde bulundurularak gerçekleştirilir. Bu çalışmada zararlı dosyadan elde edilen byte dizileri ile yeni bir sınıflandırma algoritması kullanılmıştır.

Dinamik analiz yöntemi zararlı yazılımın davranışını incelediği için zararlı dosyanın güvenli bir ortamda çalıştırılması gerekmektedir. Dinamik analiz aşamasında ağ bağlantıları, dosya sistemi işlemleri, sistem üzerindeki aktif işlemler, vb. objeler takip edilir. Dinamik analizi gerçekleştirebilmek amacıyla çalışma kapsamında DMA adında bir araç geliştirilmiştir. Kullanıcı etkileşimine sahip ve kullanımı oldukça basit olan uygulama Windows işletim sistemi üzerinde sistem tray ikon olarak çalışabilmekte ve her hangi bir zararlı aktivite tespit etmesi durumunda kullanıcıyı uyarma yeteneğine sahiptir.

Günümüzde zararlı yazılımların dinamik analiz yöntemi ile tespit edilmesinin önüne geçmek için sanal makina ve hata ayıklayıcı (debugger) tespit eden yöntemler kullanılmaktadır. Çalışmada zararlı yazılımların Vmware sanallaştırma ortamında çalıştırılıp incelenmesi gerçekleştirilmiştir. Zararlı yazılımların sanal makina tespit yöntemini aşmak için Intel tarafından geliştirilen ve uygulamaları yönlendirebilme özelliğine sahip pin aracı kullanılmıştır. Bu araç yardımıyla sanal makina tespitinde kullanılan işlem, dosya, registry anahtarı sorgulama gibi yöntemler alt edilmekte ve zararlı yazılımın gerçek makinada çalıştığını zannetmesini sağlanmaktadır.

Davranış analizinde incelenen çalıştırılabilir dosyanın aşağıdaki özellikleri dikkate alınmıştır:

- Sistem üzerindeki ağ bağlantılarındaki değişiklikler
- Sistem üzerindeki işlemlerdeki değişiklikler
- Registry dosyasındaki değişiklikler
- Servislerde gerçekleşen değişiklikler

Dinamik analiz ile tespit edilen zararlı yazılımlar daha sonra n-gram tabanlı zararlı yazılım sınıflandırma metodu kullanılarak zararlı yazılımın ailesi belirlenmektedir. Bu yöntemde her zararlı yazılım n-gram vektörü (dosyada en fazla bulunan byte dizileri) ile ifade edilmekte ve sınıflandırma işlemi bu vektör üzerinden gerçekleştirilmektedir.

Geliştirilen zararlı yazılım tespit ve sınıflandırma modülleri testi için herkese açık Pahadus zararlı yazılım seti kullanılmıştır ve ümit verici sonuçlar elde edilmiştir. Belirtilen zararlı yazılım kümesi üzerinde %86 oranında tespit etme başarısı elde edilmiştir. Zararlı yazılım tespitinden sonra sınıflandırma yöntemi olarak n-gram tabanlı sınıflandırma metodu kullanılmıştır. Sınıflandırma yönteminin öğrenme sürecinde (training phase) BILGEM zararlı yazılım yakalama sistemlerinden (honeypot) elde edilen zararlı yazılımlar kullanılmıştır. Sınıflandırma aşamasında seçilen $n=4$ ve $L=60$ çifti için başarı oranı %92 olarak elde edilmiştir.

1 Introduction

In the recent years the information security incidents have been increasing rapidly [1]. Nowadays the first targets of attackers become well-known industry companies like Sony [2], or government agencies such as Turkish Information and Communications Authority [3] and Iranian Nuclear Facilities [4]. In these security incidents malicious softwares; any software that causes harm to a computer, network, firm or especially user, play major roles.

Since the malwares and anti-malware solutions adopt entirely different purposes, there is an endless war between malware authors (writers) and malware analyzers. In this war both part advance methods to overwhelm each other. Malware authors try to prevent successful analysis by employing a variety of techniques such as anti-debugging, anti-reversing to evade detection mechanisms and prevent successful analysis. On the other hand malware analyzers search new methods to defeat malwares techniques.

To effectively fight malware, security researchers need to detect and classify them. In further step analyst must dissect malwares; learn it's stealthy and obfuscation methods, target and so on. One of the most important needs for malware analyst is the safe environment to investigate malware behavior as stated in [5]. In this thesis we created a tool which runs inside a secure virtual machine image to examine malware behavior and classified it according to our purposed statistical n-gram feature of the malware.

1.1 Thesis Organization

The rest of this thesis is organized as follows and summarized in Figure 1.1.

Chapter 2, “Prerequisites”, introduces terms, concepts and that form the prerequisites for the following chapters. The chapter clarifies the definition malware and types of malware, shows why it is crucial to analyze malware. Furthermore it introduces the malware detection methods.

Chapter 3, “Anti-virtual Machines Techniques” states the anti-virtual machine techniques to prevent to analysis attempt. The chapter include state-of-the-art techniques that malware is being run inside a virtual machine.

Chapter 4, “Implementation of the Dynamic Malware Analyzer”, describes the implementation of the dynamic malware analyzer in depth. It clarifies the question of “why do we need this kind of tool”. Strengths and limitation of the tool are also summarized in this section.

Chapter 5, “Implementation of n-gram Based Malware Classifier”, mentions the n-gram based classification method to classify the detected malware by our dynamic malware analyzer tool.

Chapter 6, “Fusion”, explains the fusion of the DMA tool and n-gram based malware classification methods to make cascade malware analyzer tool and describes accuracy of the proposed detection and classification methods.

Chapter 7, “Conclusion”, comments on the results, highlights the achievements of our work as well as inadequate part of the work. It concludes with an outlook to our future works.

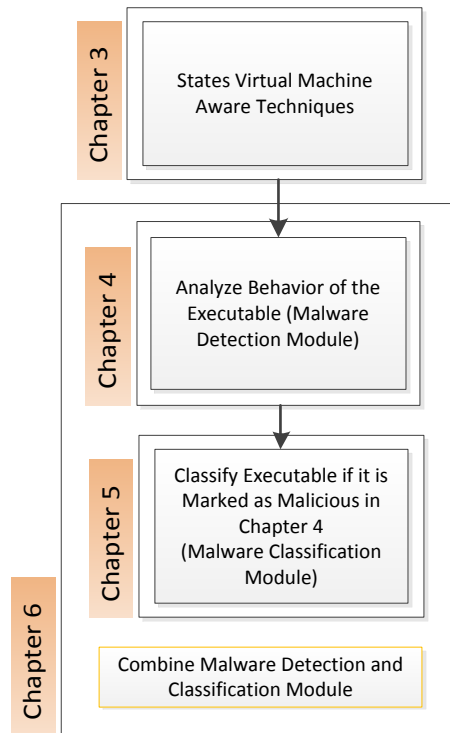


Figure 1.1. Overview of the Thesis Organization

2 State of the Art

The term malware is a conjunction of the words “malicious” and “software” and can be defined as a piece of software that is intended to perform tasks on computer systems without the users’ intention. It is designed to disrupt or deny operation, gather personal information or gain unauthorized access to system resources. This section first presents some malware-related terminology and types of malwares in section 2.1 and the next section presents the most common method that is used for malware detection.

2.1 Types of Malwares

When talking about malware you may heard virus, worm, backdoors, etc. These are the categories of the malware that is defined by the task they performed. Malware often spans multiple categories. For example, a program might have a key logger that collects passwords and a worm component that sends spam. In this section we are going to give the most common malware types that are usually used by attackers.

2.1.1 Backdoor

Malicious code that installs itself onto a computer to allow the attacker access later. Backdoors usually let the attacker connect to the computer with little or no authentication and execute command on the system. The idea has often been suggested

that computer manufacturers preinstalls backdoors on their systems to provide technical support for customers [6]. Hackers typically use backdoors to secure remote access to computer. To install backdoors, hackers may use trojan horses, worms, or other methods.

2.1.2 Trojan Horses

Trojan horse (or short trojan) is any program includes harmful or malicious payload that invites the user to run it as something normal or desirable so user may be encouraged to install it. The payload may take effect immediately and can lead to many undesirable effects or installing additional harmful software. Trojans are most commonly used for marketing. Today's advanced trojans are capable of taking complete control of web browser and possibly modify a computer's registry file.

2.1.3 Rootkits

Once a malicious program is installed on a system, it is essential that it stays concealed, to avoid detection and disinfection. The same is true when a human attacker breaks into a computer directly. Techniques known as rootkits allow this concealment, by modifying the host's operating system so that the malware is hidden from the user. Rootkits can prevent a malicious process from being visible in the system's list of processes, or keep its files from being read. Rootkits are usually paired with other malware, such as a backdoor, to allow remote access to the attacker and make the code difficult for the victim to detect.

2.1.4 Viruses

The computer virus is the most famous form of malware. It is a self-replicating program that infects a system without authorization. A virus is often transmitted via e-mail but can also be distributed through various storage mediums such as a flash drive. Once installed, it will execute itself, infect system files, and attempt to propagate to other systems. The impact of a virus ranges widely from slow system performance to wiping out every file on your computer.

2.1.5 Botnet

The term bot is short for “robot” and is used to refer to software that acts autonomously on behalf of its owner. Non-malicious bots are used by search engines to automatically index websites, or on Internet Relay Chat (IRC) to provide useful functionality to users of a particular IRC network or channel. Malicious bots typically form botnets consisting of up to several thousand infected computers. These botnets can be controlled by their owners through one or more command & control (C&C) servers, which commonly run an IRC server (or a slightly modified one) to which the bots connect and then wait for commands. Other means of communication that are employed by bots are the HTTP protocol, or peer-to-peer protocols.

Botnets can be used to launch distributed denial-of-service (DDoS) attacks, e.g., for the botnet owners' entertainment, or, to blackmail companies by threatening to attack on their critical infrastructure. Other uses of botnets are for sending illegitimate bulk e-mail, spying on infected computers and their users (e.g., stealing their authentication credentials, credit card information or other private data). More information about bots and botnets can be found in [7].

2.2 Malware Analysis Methods

Usually when performing malware analysis, you have only the malware executable, which is not human-readable. In order to make sense of the executable, there are variety of tools and tricks which reveal some information about them. These tools use two fundamental approaches to analysis malware: static and dynamic analysis. Static analysis involves examining the malware without running it. Dynamic analysis involves running malware. Both techniques have pros and cons in certain case. Static analysis helps to produce malware signatures but it is largely ineffective against armed (sophisticated) malware and it may miss important behaviors.

2.2.1 Static Analysis

Static analysis consists of examining the executable file without viewing the actual instructions. It is usually the first step in examining malware and can confirm whether a file is malicious, provide information about its functionality, and sometimes provide information that will allow you to produce simple network signatures. It is straightforward and can be quick, but it's largely ineffective against sophisticated malware, and it can miss important behaviors. By the way it is pretty safe because you do not execute the dangerous code but it is still best to undertake on an isolated machine.

In first step of static analysis consists of looking for obvious indicators as to what the attacker is. Basically the file fingerprint (usually a MD5 hash) is calculated and determined if the match is found with a known malware. The first step sounds like traditional antivirus.

In the deep file analysis, the study focus on the file format and content. The following checks are used;

File Packing: Determine if the file is packed before deep search. If it is packed try to unpacked it and obtain pure executable.

Plain Text Matching: Look for the plain text of the executable and obtain as much information as possible. Generally *strings* utility is used to explorer plain text in the file.

Disassembly: The last static analysis technique is disassembly. In this step the disassembly is used to examine the machine code of the executable and step thorough it (as in a debugger) to figure out exactly what the program is doing. This is pretty advanced method, but it may not reveal all the details of what the malware does unless you won't execute it.

2.2.2 Dynamic Analysis

Dynamic analysis techniques involve running the malware and observe its behavior on the system. These techniques can give valuable information that is difficult to obtain with other techniques. Like static analysis techniques, dynamic analysis techniques won't be effective with all types of malware and can miss important functionality because of anti-dynamic analysis techniques such as anti-virtual machine, anti-debugger and so on. Therefore the malware analyst must give attention to these types of tricks.

Malware typically can change all sorts of things on the compromised device, so dynamic analysis consists of monitoring the followings;

Volatile Memory: Malware can overflow buffers and use this memory location to gain access to the device. By capturing and analyzing the device memory, it is possible to figure out whether and how the malware uses memory.

Registry/Configuration Changes: Look for any evidence of the registry changes when performing dynamic analysis because malware often changes registry values to gain persistent access to the system.

File Activity: Malwares may also add, alter or delete files. So by monitoring file activity the analyst obtains valuable information about the behavior of the suspicious file.

Processes/Services: Look for new or stopped processes or services because a lot of malware shuts down AV engines, or install new services to obtain persistent access to system.

Network Connection: The network connection monitoring is essential part of dynamic analysis to understand what malware is doing. We can obtain the malware's destination IP address, port and protocol that are used by malware to communicate with compromised system.

3 Anti-Virtual Machine Techniques

Malware authors sometimes use anti-virtual machine (anti-VM) techniques to prevent from analysis attempts. With these techniques, the malware attempts to detect whether it is being run inside a virtual machine. If a virtual machine is detected, it can act differently or simply do not run. This can, of course, cause problems for the analyst.

Today both system administrators and users use virtual machines in order to make it easy to rebuild a machine from a snapshot. Since malware authors realized that virtualization technology is used to dissect malicious executable, they started to obfuscate their source with the anti-virtual machine tricks. Because anti-VM techniques typically target VMware, in this chapter we'll focus on anti-VMware techniques.

3.1 Hardware Fingerprinting

Hardware fingerprinting involves looking for special virtualized hardware pattern unique to virtual machine. For example the MAC address of the network card, specific hardware controllers, BIOS, graphic card, etc. Table 3.1 shows the results of hardware fingerprinting which obtained on a host and on a guest OS running on VMware. This fingerprinting carried out using Windows Management Instrumentation (WMI) classes and APIs [8].

Table 3.1. Hardware Fingerprinting of Native and VMware Machine, [12]

Hardware Component	Attribute	VMware Machine	Native Machine
Motherboard	Serial No	-	.2GTP3BS.CN7016697MG1DN.
SCSI Controller	Caption	VMware SCSI Controller	Microsoft iSCSI Initiator
BIOS	Serial Number	VMware-56 4d 68 4c f9 e5 62 f4-fb 4d f0 5b 88 28 29 d9	2GTP3BS
USB Controller	Caption	1. Intel(R) 2371AB/EB PCI to USB Universal Host Controller 2. Standard Enhanced PCI to USB Host Controller	1. Intel(R) ICH9 Family USB Universal Host Controller – 2936 2. Intel(R) ICH9 Family USB Universal Host Controller – 2938 3. Intel(R) ICH9 Family USB Universal Host Controller – 2937
Network Adapter	Caption	VMware Accelerated AMD PCNet Adapter	1. WAN Miniport (SSTP) 2. WAN Miniport (IKEv2) 3. WAN Miniport (L2TP)
Network Adapter	Mac Address	00:0C:29:28:29:D9 (This MAC address falls in VMware MAC Address Range)	50:50:54:50:30:30

3.2 Registry Check

The registry is a centralized, hierarchical database for application and system configuration information in Windows operating system. Access to the registry is through registry keys, which are analogous to file system directories. A key can contain other keys or key/value pairs, where the key/value pairs are analogous to directory names and file names. Each value under a key has a name, and for each key/value pair, corresponding data can be accessed and modified.

The user or administrator can view and edit the registry contents through the registry editor, for example using the *regedit* command. Alternatively, programs can manage the registry through the registry Windows API functions. UNIX systems store similar information in the directory and files in the user's home directory. The registry centralizes all this information in a uniform way.

The Figure 3.1 shows a typical view from the registry editor and gives an idea of the registry structure and contents.

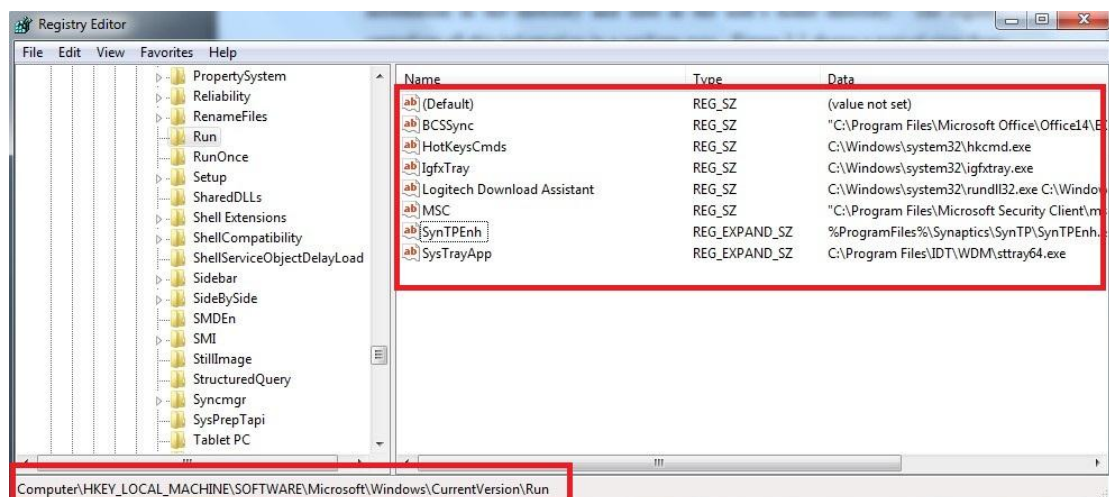


Figure 3.1. Startup Programs Obtained from Registry

The registry contains information such as the following and is stored hierarchically in key/value pairs;

- Windows version number, build number, and registered users
- Similar information for every properly installed application

- Information about the computer's processor type, number of processors, memory, and so on
- Security information such as user account names
- Installed services

Tobias Klein's tool ScoopyNG [9] includes a small code that looks for certain keys within the Windows registry to determine that if the machine is virtual.

3.3 Memory Check

This technique involves looking at the values of specific memory locations after the execution of instructions such as SIDT (Store Interrupt Descriptor Table), SLDT (Store Local Descriptor Table), SGDT (Store Global Descriptor Table), and STR (Store Task Register) [9] [10] [11]. It is the most widespread detection technique employed by the present VM detecting malware [12].

RedPill, discovered by *Joanna Rutkowska*, is based on checking the Interrupt Descriptor Table (IDT). More info on this can be obtained from Joanna's web page [13], and in [11]. Both techniques are based on the simple fact that any machine, virtual or not, will need its own instance of some registers. Systems such as VMware will create dedicated registers for each virtual machine. These registers will have a different address than the one used by the host system, and by checking the value of this address, the virtual system's presence can be detected. Code samples can be seen Figure 3.2.

Besides that, *Red Pill* succeeds only on a single-processor machine. It won't work consistently against multicore processors because each processor (guest or host) has an

IDT assigned to it. Therefore, the result of the SIDT instruction can vary, and the signature used by *Red Pill* can be unreliable.

```
int swallow_redpill() {
    unsigned char m[2+4],rpill[]="0f010d0000000000c3";
    *((unsigned*)&rpill[3])=(unsigned)m;
    ((void(*)())&rpill)();
    return (m[5]>0xd0)?1:0;
}
```

Figure 3.2. Snap Code of Red Pill Technique

3.4 VMware Communication Channel Check

Ken Kato discovered the presence of a host-guest communication channel so called backdoor Input/Output (I/O) port [14]. VMware uses the I/O port 0x5658 ('VX' in ASCII) to communicate with the host machine. It is obvious this port is not real. The verification is as follows:

1. The magic number 0x564D5868 ('VMXh' in ASCII) is loaded in the EAX register.
2. The proper parameter of the command that is to be sent is loaded in EBX register.
3. The command to be used is loaded in the ECX register. For example, the command 0x0A brings back the VMware version.
4. It is read from 'VX' port. If we have 'VMXh' in the EBX register, this means that we are under VMware.

There are more commands supported by the backdoor I/O port; for example to obtain data from the Windows clipboard or the speed in MHz of the microprocessor. The most important commands are displayed in Figure 3.3. A detailed documentation can be found on the VM Back website [14].

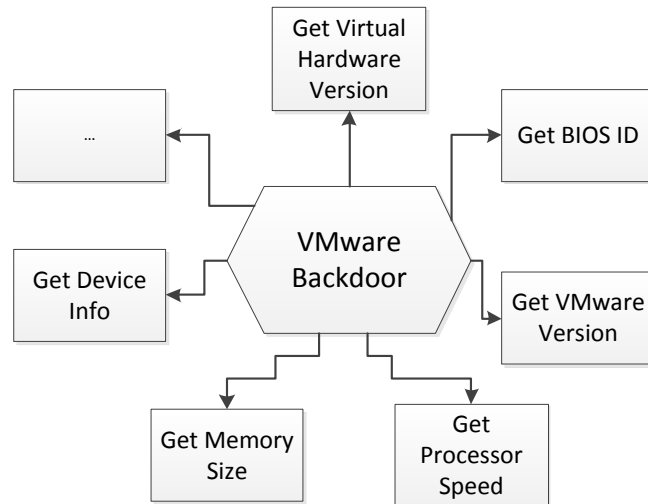


Figure 3.3. VMware I/O Backdoor's Main Functionalities

```

MOV EAX, 564D5868h; 'VMXh' magic number
MOV ECX,0Ah ; get VMware version command-specific-parameter
MOV DX, 5658h ; 'VX' backdoor-command-number
MOV DX, 5658h; VMware I/O Port
IN EAX, DX; "returns" version number in EAX
  
```

Figure 3.4. Assembly Code to Detect VMware Machine via VMware I/O Port

3.5 File & Process Check

The VMware environment creates many artifacts on the system, especially when VMware Tools is installed. There are many VMware specific processes such as VMwareUser.exe, vmacthlp.exe, VMwareService.exe, VMwareTray.exe that constantly run in the background. There also exist some VMware specific files and folders. Hence querying for these objects could also serve as a method for VM detection.

Malware can use these artifacts, which are present in the file system and process listing, to detect VMware. For example, Figure 3.5 shows the process listing for a standard VMware image with VMware Tools installed. Notice that three VMware processes are running: VMwareService.exe, VMwareTray.exe, and VMwareUser.exe. Any one of these can be found by malware as it searches the process listing for the VMware string.

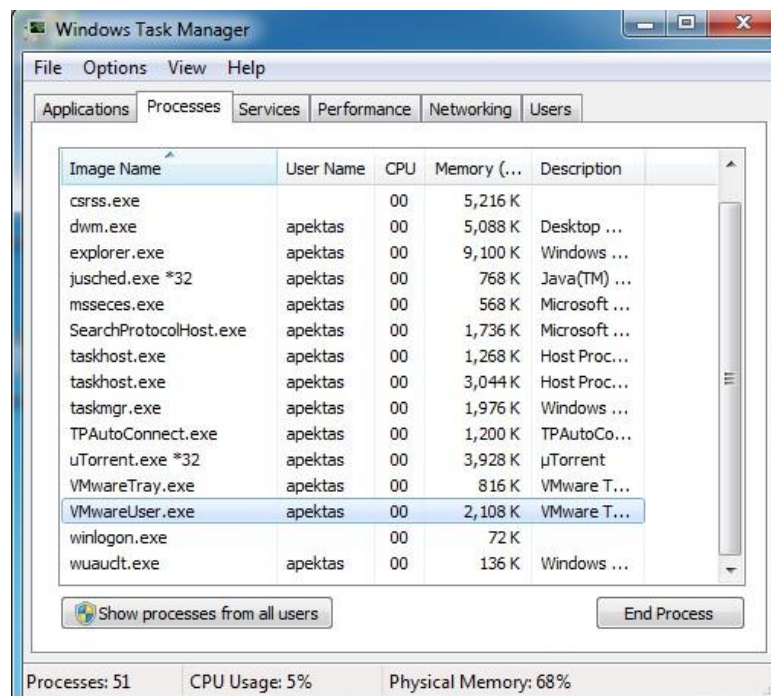
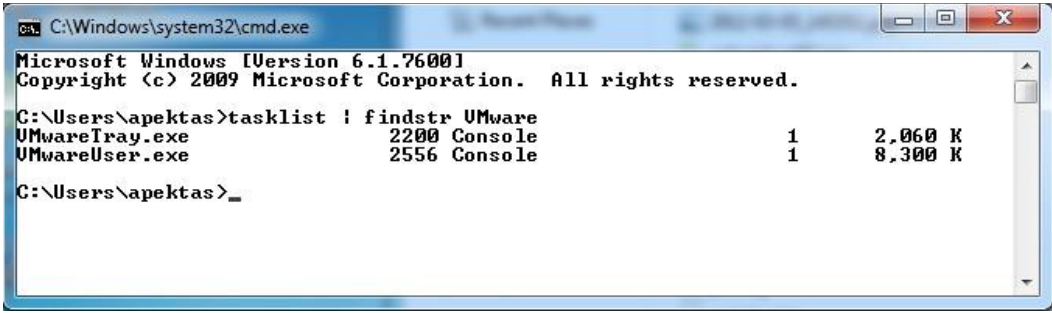


Image Name	User Name	CPU	Memory (...)	Description
csrss.exe		00	5,216 K	
dwm.exe	apektas	00	5,088 K	Desktop ...
explorer.exe	apektas	00	9,100 K	Windows ...
jusched.exe *32	apektas	00	768 K	Java(TM) ...
msseces.exe	apektas	00	568 K	Microsoft ...
SearchProtocolHost.exe	apektas	00	1,736 K	Microsoft ...
taskhost.exe	apektas	00	1,268 K	Host Proc...
taskhost.exe	apektas	00	3,044 K	Host Proc...
taskmgr.exe	apektas	00	1,976 K	Windows ...
TPAutoConnect.exe	apektas	00	1,200 K	TPAutoCo...
uTorrent.exe *32	apektas	00	3,928 K	µTorrent
VMwareTray.exe	apektas	00	816 K	VMware T...
VMwareUser.exe	apektas	00	2,108 K	VMware T...
winlogon.exe		00	72 K	
wuaudt.exe	apektas	00	136 K	Windows ...

Processes: 51 CPU Usage: 5% Physical Memory: 68%

Figure 3.5. Process Listing on a VMware Image

VMwareService.exe runs the VMware Tools Service as a child of services.exe. It can also be identified by searching the registry for services installed on a machine or by listing services using the “**tasklist**” or “**net start**” command (Look at Figure 3.6).



```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\apektas>tasklist /findstr VMware
VMwareTray.exe           1           2,060 K
VMwareUser.exe          1           8,300 K

C:\Users\apektas>_
```

Figure 3.6. Search VMware Processes in Process List

The VMware installation directory (default path C:\Program Files\VMware\VMware Tools) may also contain artifacts. A quick search for “VMware” in a virtual machine’s file system might find clues about existing of the VMware image.

4 Implementation of Dynamic Malware Analyzer

We have developed an application called “Dynamic Malware Analyzer” in order to analyze malicious software by running on safe virtualized environment. .Net framework was used to create DMA. It can monitor anomalies occurred on the system through checking out all processes, connection table and service details on Windows operating system. DMA has a user-friendly graphical user interface, can be used easily and efficiently in dynamic analysis by malware researchers. Before running DMA you have to configure some simple settings.

In this section, firstly we will present our methods used to bypass the countermeasures taken by malware (known as anti-virtual machine aware malware or split personality malware) to run itself on the virtual machine. Then, the actions taken to monitor changes occur on the system will be explained. To evaluate the accuracy of the developed application we tested it over the samples obtained from [15]. Finally, we will cover the pros and cons of our dynamic malware analyzer tool.

4.1 Binary Instrumentation Tool: Pin

Pin is a free tool provided by Intel for the dynamic instrumentation of programs, i.e. arbitrary code (written in C or C++) can be injected at arbitrary places in the executable. It supports Linux binary executables for IA-32, Intel64 (64 bit x86), and Itanium (R) processors; Windows executables for IA-32 and Intel64; and MacOS executables for IA-32.

Pin does not instrument an executable statically by rewriting it, but rather adds the code dynamically while the executable is running. This also makes it possible to attach Pin to an already running process.

Pin come up with the source code for a large number of example instrumentation tools. If you look at the source code of these examples you can easily understand that it is easy to derive new tools using the examples as a template.

4.1.1 Intended use of Pin

As indicated in “Anti-Virtual Machine Techniques” section, some malicious software has special controls as listed follows to check if it is running on the virtual machine and if it detects virtual machine it modifies behavior, acts harmless or simply not run.

- Hardware fingerprinting
- Registry lookup
- Memory lookup
- VMware communication lookup
- File & Process lookup

In this study we have used pin tool provided by Intel to detect and execute anti-VM aware malwares on the VMware. Figure 4.1 illustrates the step by step with pseudo-code.

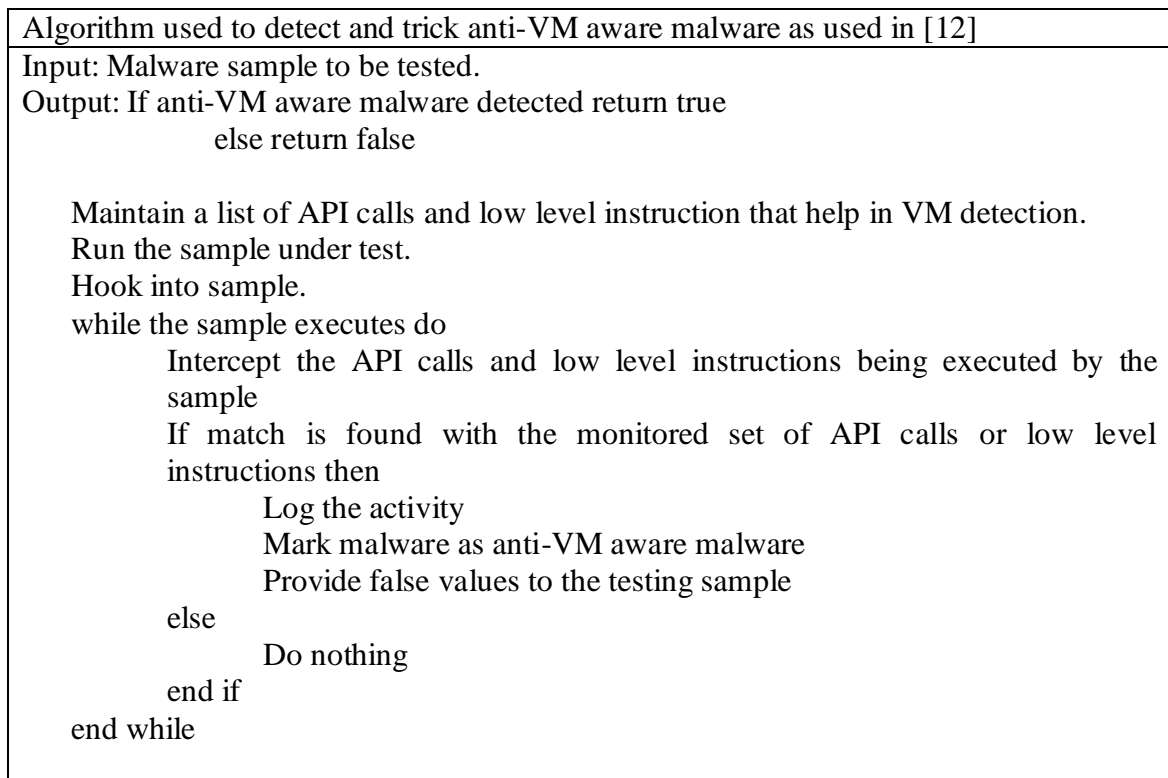


Figure 4.1. Our Approach for Detecting anti-VM Aware Malware

Maintained list of Windows API calls, split into two categories; hardware and registry APIs, listed as follows to detect anti-VM aware malware.

1. Hardware Querying API list
 - a. SetupDiEnumDeviceInfo()
 - b. SetupDiGetDeviceInstanceId()
 - c. SetupDiGetDeviceRegistryProperty()
 - d. WMI APIs

2. Registry Querying API list
 - a. RegEnumKey()
 - b. RegEnumValue()
 - c. RegOpenKey()
 - d. RegQueryInfoKeyValue()

- e. RegQueryMultipleValues()
- f. RegQueryValue()

Let us consider the example of a sample that makes the following API calls with the given arguments:

- RegOpenKeyEx(

```
HKEY_LOCAL_MACHINE,
TEXT ("HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi Bus
0\\Target Id 0\\Logical Unit Id 0"), 0,
KEY_QUERY_VALUE,
&hKey);
```

- RegQueryValueEx(

```
HKey,
TEXT ("Identifier"),
NULL, NULL,
(LPBYTE) PerfData,
&cbData);
```

In the above cases, the key value returned in a VMware machine will contain the string “VMware”. Thus, we monitor the values returned by the OS in response to the API calls made by the sample. If it contains the string “VMware”, the control passes to our replacement routine where we change the value to a more appropriate value such as “Microsoft” or to a value that would have been returned on a host Windows OS.

Similarly when VM specific instructions such as SIDT are in the course of being executed by the sample, the control passes to our replacement routine where we set the value of the destination operand to a value that would be obtained on the host Windows OS.

4.2 Monitor System

In dynamic analysis methods, monitoring the usage of the system resources is an important topic that must be taken into account by an analyst. Process handle details and connection table are two major concepts in the system that is used to detect and analyze malware. Therefore DMA has four main monitoring capabilities listed as follows in order to monitor system details;

1. Monitoring Processes
2. Monitoring Connections
3. Monitoring Services
4. Monitoring Register (comparison based monitoring)

4.2.1 Monitor Processes

Processes are the essential building blocks of any Microsoft Windows system. Knowing what processes are active on the system at any given time can help you understand how system resources are being used, and it can assist you in diagnosing problems and identifying malware.

All windows versions were shipped with *Task Manager* to provide users insight into process activity for viewing the processes (application and services) that are running on the system. To avoid confusing users, *Task Manager* provides limited information about the processes.

Task Manager is the application that users usually used it in order to find out why their system is slow and to kill degenerate processes. It often doesn't give deep enough knowledge about what is causing a process to misbehave, nor does it provide key data that can help a technical user to identify a process as malware.

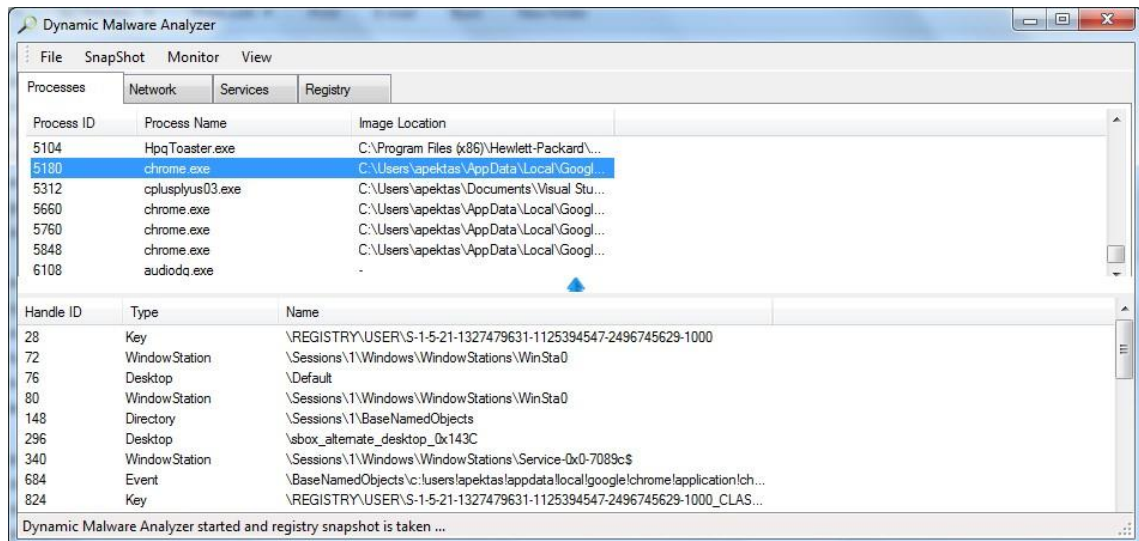


Figure 4.2. Process Monitoring Feature of DMA

DMA tool likes standard Task Manager on windows and its purpose is to evaluate the newly created and the state-changing processes. In addition to this feature the location of the running application on the file system can be easily obtained from the “Image Location” title of the Process tab. Even though users with limited privileges can see all running processes, they can't see all information about the processes such as location of the executable and handles of the process.

If the user clicks on an active process, the advanced features of the process like threads, registry key handles, file handles, and etc. appears on the bottom side of the application. As you guess these advanced features don't show to the limited privileged user. The

user also has the ability to kill the process that he/she wants to end within his/her privilege level.

4.2.2 Monitor Connections

As malware often try to communicate with a command-and-control server to manage the system remotely we take into account the network connections. The following basic attributes of network activity;

1. Destination IP address
2. TCP and UDP port (TCP/IP transport layer protocols)
3. Domain Names
4. Traffic Content

are used extensively by security analysts to ensure defense. DMA is capable of keeping an eye on networking events. On other words, DMA can display and log the list of all currently opened TCP/IP ports on the system. For each connection on the list, information about that open connection is also displayed, including the process name.

In addition, just as described in “Process Monitor” section if the user clicks on an active connection, the handlers of this connection appear on the bottom side of the application. On the other hand, DMA allows the users to close the suspicious connection by killing the related process.

Process ID	Process Name	Protocol	Remote Host	Remote Port	Local Host	Local Port	Status
3952	Dropbox.exe	TCP	199.47.216.144	80	192.168.1.2	22108	ESTABLISHED
5180	chrome.exe	TCP	80.239.230.153	80	192.168.1.2	26731	ESTABLISHED
5180	chrome.exe	TCP	80.239.230.168	80	192.168.1.2	26710	ESTABLISHED
5180	chrome.exe	TCP	80.239.230.168	80	192.168.1.2	26711	ESTABLISHED
5180	chrome.exe	TCP	80.239.230.168	80	192.168.1.2	26722	ESTABLISHED
5180	chrome.exe	TCP	80.239.230.168	80	192.168.1.2	26723	ESTABLISHED
3952	Dropbox.exe	TCP	199.47.217.172	443	192.168.1.2	26010	CLOSE-WAIT
3952	Dropbox.exe	TCP	199.47.217.172	443	192.168.1.2	26011	CLOSE-WAIT
3952	Dropbox.exe	TCP	199.47.217.177	443	192.168.1.2	22101	CLOSE-WAIT
5180	chrome.exe	TCP	65.55.58.199	80	192.168.1.2	26734	ESTABLISHED
5180	chrome.exe	TCP	208.117.239.2...	80	192.168.1.2	26780	ESTABLISHED
5180	chrome.exe	TCP	65.55.11.240	80	192.168.1.2	26708	ESTABLISHED
5180	chrome.exe	TCP	207.249.186.82	443	192.168.1.2	26866	SYN-SENT

Handle ID	Type	Name
76	WindowStation	\Sessions\1\Windows\WindowStations\WinSta0
80	Desktop	\Default
84	WindowStation	\Sessions\1\Windows\WindowStations\WinSta0
116	Key	\REGISTRY\USER\S-1-5-21-1327479631-1125394547-2496745629-1000
164	Directory	\Sessions\1\BaseNamedObjects
832	Mutant	\Sessions\1\BaseNamedObjects\dropbox_95848a61
856	Key	\REGISTRY\USER\S-1-5-21-1327479631-1125394547-2496745629-1000_CLAS...
1844	Key	\REGISTRY\USER\S-1-5-21-1327479631-1125394547-2496745629-1000_CLAS...

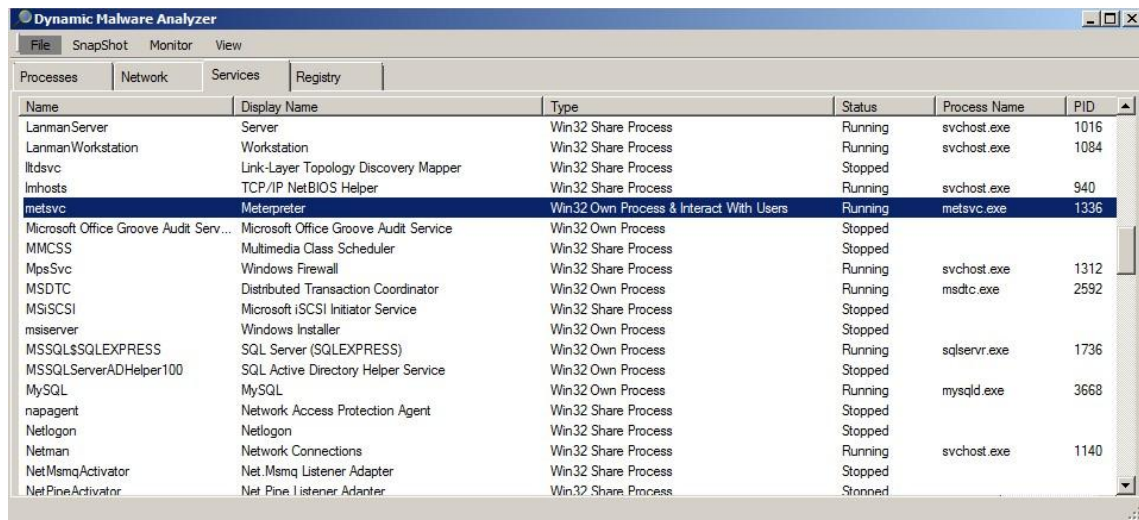
Figure 4.3. Network Monitoring Feature of DMA

4.2.3 Monitor Services

Windows services are critical processes that provide server functionality, such as Active Directory, e-mail, DNS, automatic updates, etc. One example is Microsoft Exchange Server, which has several services, such as Information Store and SMTP; the failure of even one of these services results in suspended e-mail delivery or even lost messages until the services are started again.

After going through all the hard work of exploiting a system, malware often create services to maintain access to system. This way, if the connection breaks, you can still gain access to the system. Thus, DMA is capable of monitoring changes occurred in Windows services.

Figure 4.4 illustrates that an attacker installed *Meterpreter* as a service on the exploited system in order to access the victim box later and can proceed with further recognition, enumeration and exploitation for the specific system and/or the network.



Name	Display Name	Type	Status	Process Name	PID
LanmanServer	Server	Win32 Share Process	Running	svchost.exe	1016
LanmanWorkstation	Workstation	Win32 Share Process	Running	svchost.exe	1084
ltdsvcs	Link-Layer Topology Discovery Mapper	Win32 Share Process	Stopped		
lmhosts	TCP/IP NetBIOS Helper	Win32 Share Process	Running	svchost.exe	940
metsvc	Meterpreter	Win32 Own Process & Interact With Users	Running	metsvc.exe	1336
Microsoft Office Groove Audit Serv...	Microsoft Office Groove Audit Service	Win32 Own Process	Stopped		
MMCSS	Multimedia Class Scheduler	Win32 Share Process	Stopped		
MpsSvc	Windows Firewall	Win32 Share Process	Running	svchost.exe	1312
MSDTC	Distributed Transaction Coordinator	Win32 Own Process	Running	msdtc.exe	2592
MSISCSI	Microsoft iSCSI Initiator Service	Win32 Share Process	Stopped		
msiserver	Windows Installer	Win32 Own Process	Stopped		
MSSQL\$SQLEXPRESS	SQL Server (SQLEXPRESS)	Win32 Own Process	Running	sqlservr.exe	1736
MSSQLServerADHelper100	SQL Active Directory Helper Service	Win32 Own Process	Stopped		
MySQL	MySQL	Win32 Own Process	Running	mysqld.exe	3668
napagent	Network Access Protection Agent	Win32 Share Process	Stopped		
Netlogon	Netlogon	Win32 Share Process	Stopped		
Netman	Network Connections	Win32 Share Process	Running	svchost.exe	1140
NetMsmqActivator	Net.Msmq Listener Adapter	Win32 Share Process	Stopped		
NetPineActivator	Net.Pine.Listener.Adapter	Win32 Share Process	Stopped		

Figure 4.4. Service Monitoring Feature of DMA

4.2.4 Monitor Registry

Malware often uses the registry for persistence or configuration data. The malware adds entries into the registry that will allow it to run automatically when computer reboots. The registry is so large that there are many ways for malware to use it for persistence. It seems like more and more programs are attempting to install spyware, advertisements, or other garbage without your knowledge.

To detect registry changes we inspired open source Regshot [16] that the program will create snapshot of the actual states of registry. This means that at any time you can open the snap file and view the contents of the registry just as they were when you've scanned

the registry. The program can compare two different snapshot files so you'll see exactly what changed in the registry between the two scans.

Figure 4.5 displays a subset of the results generated by DMA during malware analysis. Registry snapshots were taken before and after running the “Lab07_01.exe” malware obtained from [5].

As you can see a new service is installed on the system, you can check it by looking at services and registry tables on the compromised system. Sample malware added “*HKLM\SYSTEM\ControlSet001\services\Malservice\Start*” key and set the start key as 2 (means autostart) to start malware automatically when system reboots.

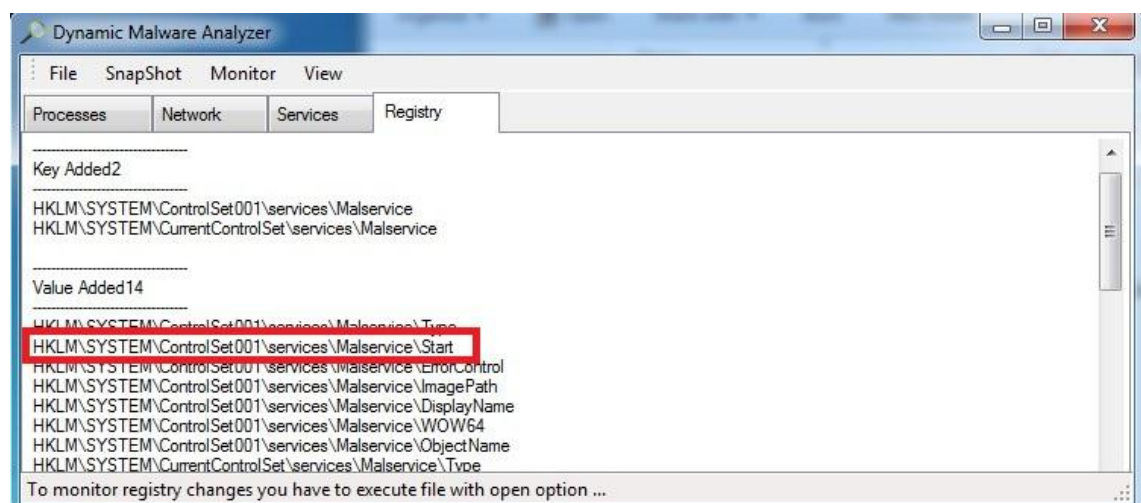


Figure 4.5. An Example of Registry Changes Made by Malware to Gain Persistence

4.3 Basic Configuration

To work properly with DMA there are a few settings to be set. These requirements are discussed in this chapter. First of all, we have to set the exact location of the pin on the file system. Then we select the possible application that may create network connection and finally we restrict the destination point with the aid of public malware domain lists published at regular intervals by [17] and [18].

4.3.1 Restrict Connectable Process

The application which is not determined as a connectable application by user and creates network connection is reported by DMA while informing the users.

These mentioned processes can be decided by the following two ways:

1. When the user wants all the applications except active application not to make network connection he/she can snapshot application meaning that the user determines the connectable applications.
2. User may save the applications which can make connection, into a file called “connectable.txt”

If an application that is not in determined list establishes network connection, the DMA informs the users and log the activities as seen Figure 4.6. In this example, a malicious executable called “custom.exe” which is not in the connectable application list is trying to establish connection in order to control the victim host machine remotely.

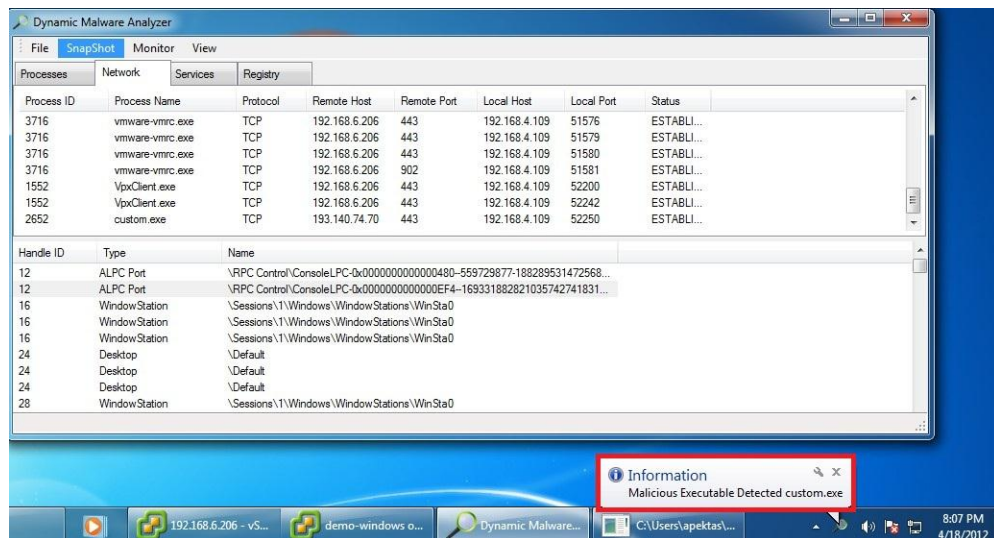


Figure 4.6. A Malicious File Establishes Connection without User's Intent

4.3.2 Restrict Destination Domain

DMA has also the ability to filter network connection by checking the destination domain or IP. If the DMA detects that system establishes connection to the malicious domain, DMA logs the malicious connection and informs users via popups. In the study we used the malicious domains collected by [17] and [18] to restrict the destination points. These domain lists are live list and updated frequently. To derive IP details from domain information we used the following Ruby script:

```

require 'resolv'
file=File.open("malwareips.txt","w")
IO.foreach("malware_domainlist.txt") do |line|
  begin
    array=Resolv.getaddresses(line.strip)
    if(array.length>0)
      array.each {|element| puts element; file.puts element}
    end
  rescue Exception
    puts ""
  end
end
end

```

Figure 4.7 DNS Resolver Script for Malware Domains

Derived IP list must be saved a file which is called “Domainlist.txt” that locates in the same directory of the DMA. After all these procedures, if an application tries to connect to a specified malicious destination, as shown in Figure 4.8 DMA produces a warning message and the user is notified. In this example, chrome.exe, attempting to connect a malicious target which exists in the “Domainlist.txt” file, is detected and the user is notified by DMA.

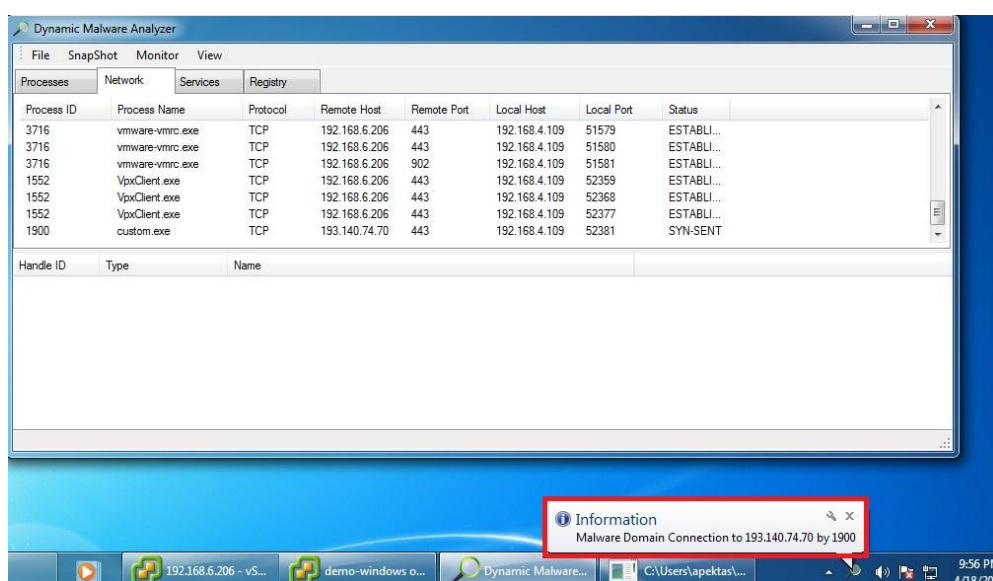


Figure 4.8. A Malicious Process Tries to Connect Malicious Domain

4.4 Comparison of Dynamic Analysis Techniques

In this chapter, firstly we will talk about the existing dynamic malware analysis methods. Then we will mention the existing methods and DMA tool and finally we will cite the future works and pros & cons of DMA.

4.4.1 Overviewing Existing Dynamic Analysis Techniques

Currently dynamic malware analysis is a very popular research topic. Several methods exist for automatically analyzing malicious software behaviors. These can be gathered into two groups [19]:

1. Analysis the difference between two snapshots of the system, one taken before the malware execution, the other after.
2. Monitor the actions performed by malware during its execution.

API hooking is mostly used technique in order to trace the behavior of the malware and though this technique it can be possible to get a control flow of the executable. Most dynamic analysis framework such as CWSandbox [20], BitBlaze [21] and TTANalyze [22] use this technique. The concept of hooking is simple, each time an application accesses an API function it gets sent to different location, where the modified code is located.

API hooking has to be done in a careful way in order to be transparent and undetectable for the malware. It is never good if a malware detects that it is running in a simulated environment as then it is modifying its behavior. This has been stated in previous chapters.

A way to bypass the API hooking trap would be call the kernel functions directly and avoid the usage of the API, anyhow this is uncommon as for this the programmer has to know exactly what version of operating system and on what service pack patch level.

As the goal of most malware is to infect a huge user base and not only targeted persons, directly kernel function calling is not easy to realize.

Another popular method used in dynamic analysis is DLL (Dynamic Link Library) injection. This can be realized through API hooking with inline code overwriting. Therefore the applications have to be patched once it has been loaded into the memory. The address space of the malware has to contain our hooked function in order to be able to call the hook from inside the malware's address space. This technique is realized by a specialize thread located into the malware's memory allocation. CWSandbox [20] framework uses this technique to make the malicious code load their DLL in their address space.

A further technique that is widely used is Virtual Machine Introspection (VMI). This technique allows a monitoring of a virtual machine without risk. This method allows us to have both advantages, a good resistance against attacks on the one hand, and full control of what is happening in the host on the other hand. Therefore the VMI uses access to the hardware-level state, for example the state of the physical memory pages and registers and also events like memory accesses and interrupts. The knowledge of these events and states allows us to map the events to OS-level semantics. For example livewire [23] implement this technique to detect intrusion.

To remind, there has to be said that malware can implement some functionalities to detect if it is running on a virtual machine. If this is the case the malware can adapt its behavior. One famous project implementing this detection functionality is the *Red Pill* project of Joanna Rutkowska. [11]

4.4.2 Pros & Cons of Our Dynamic Analysis Techniques

First of all, unlike other dynamic malware analysis methods our developed application can overcome anti-VM techniques described in 3. Chapter, thus in this perspective it is significantly different from other methods. Although there exists studies to detect anti-VM aware malware [24] [25] [26] [27], we couldn't find the work that examines anti-VM aware malware behavior by running it. Therefore our method offers an advantage of dynamic analysis of anti-VM aware malwares.

Since the application has user-friendly interface and works as a system tray in the background, the application don't interfere the user. If there is an anomaly DMA logs the event with a sufficient detail and informs the user about the situation. DMA can be run on any system that is installed .Net framework 3.0 and it doesn't require any extra installation and configuration except the requirements stated in Chapter 4.3.

DMA, can be used as an anti-malware software, can reduce dependency on the anti-virus software and make to feel more safe the users since as stated before anti-virus solution is based on signature and can be bypassed easily.

DMA is currently still under development stage so it can't handle all the tasks automatically. For example suppose that you want to analyze a set of malicious software. If the sample in the set is detected as malware and make same changes to system then you have to come back the original image via snapshot technology in the virtual machine. This steps cause too much workload. Therefore, it is clearly apparent that DMA have to automate malware analysis in order to carry out the dynamic analysis effectively

On the other hand while detecting the malware samples, there may be scoring mechanism to describe the harmfulness degree of the file. The realization of the scoring mechanism will provide fast, more reliable, less struggle and user-independent analysis.

5 Implementation of n-gram Based Malware Classifier

In this section firstly we will give the definition of n-gram and usage fields in computer science. After then the existing n-gram approaches is explained we will propose our n-gram based malware classification methodology. Finally we will elaborate and compute the accuracy of the proposed methodology.

5.1 Definition of n-gram

N-gram is a fixed size sliding window of byte array, where n is the size of the window. For example the “558D6C24948B45...400085C0” sequence is segmented (represented) into 4-gram as “558D6C24”, “8D6C2494”, “6C24948B” and “24948B45” as seen in the Figure 5.1.

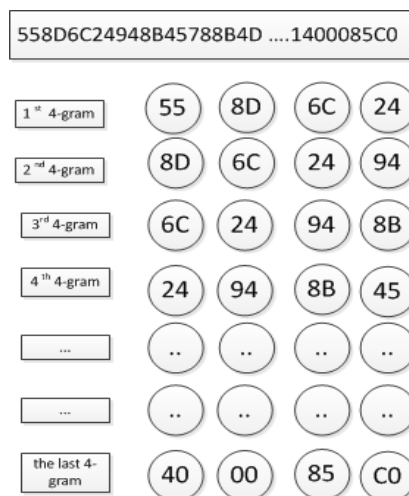


Figure 5.1. N-gram Sequences

An n-gram of size 1 is referred to as a “unigram”; size 2 is a "bigram" (a.k.a “digram”); size 3 is a “trigram”. Larger sizes are sometimes referred to by the value of n, e.g., “four-gram”, “five-gram”, and so on.

N-gram is used in different fields such as natural language processing, authorship detection, information gathering and also malware detection. They have been used in the following application;

- Find likely candidates of misspelled words
- Improve compression in compression algorithm
- Improve accuracy of the speech recognition
- Improve performance of the information retrieval process
- ...

5.2 Existing n-gram Approaches to Analyze Malware

The representation of malware by using n-gram profiles has been presented in the open literature; see for example [28], [29] and [30]. In these studies some promising results towards malware detection are presented. However malware domain has been evolving due to survivability requirements.

Malware has to evade anti-virus scanners to perform its functions. Obfuscation techniques have been developed in order to avoid detection by antivirus scanner. And these techniques disturb n-gram features of binary form of the malware used by the previous work. Similar methodologies have been used in source authorship, information retrieval and natural language processing [31], [32].

The first known use of machine learning in malware detection is presented by the work of *Tesauro et al.* in [33]. This detection algorithm was successfully implemented in IBM's antivirus scanner. They used 3-grams as a feature set and neural networks as a classification model. When the 3-grams parameter is selected, the number of all n-gram features becomes 256^3 , which leads to some spacing complexities. Features are eliminated in three steps: first 3-grams in seen viral boot sectors are sampled, then the features found in legitimate boot sectors are eliminated, and finally features are eliminated such that each viral boot sectors contained at least four features. Size of feature vectors in n-grams based detection models becomes very large so feature elimination is very important in these models. The presented work has been limited by the boot sector viruses' detection because boot sectors are only 512 bytes and performance of technique is degraded significantly for larger size files.

As a historical track, IBM T.J. Watson lab extended boot virus sector study to win32 viruses in 2000 [34]. At this stage, 3 and 4 grams were selected and encrypted data portions within both clean files and viral parts were excluded due to the fact that encryption may lead to random byte sequences. At the first instance, n-grams existed in constant viral parts were selected as features and then, the ones existed in clean files more than a given threshold value were removed from the feature list. In this study, along the use of neural networks boosting was also performed. Results of this study showed that the developed method performance was not sufficient. Schultz *et al.* has used machine learning methods in [35]. Function calls, strings and byte sequence were used as the feature sets. Several machine learning methods such as RIPPER, Naive Bayes and Multi Naive Bayes were applied, the highest accuracy of 97.6% with Multi Naive Bayes was achieved.

Abou-Assaleh et al. [29] contributed to the ongoing research while using common n-gram profiles. k nearest neighbor algorithm with k=1 instead of the other learners was used. Feature set was constituted by using the n-grams and the occurrence frequency, where the occurrence frequency is denoted by L. Tests have been done with different n

(ranging from 1 to 10) and L (ranging from 20 to 5000) values. Data set used in these experiments was kept fairly conservative of 25 malware and 40 benign files. With this set, test results shown 98% of success. Using the data in [29], the accuracy slightly dropped to the 94% level.

Kolter et al. [28] used 4-grams as features and selected top 500 n-grams through information gain measure. They used instance based learners, TFIDF, naive bayes, support vector machines, and decision trees and also boosted last three learners. Boosted decision tree outperformed all others and gave promising results such as ROC curve of 0.996.

While the battle between malware authors and anti-virus producers are continuing, our motivation is to find the statistical method to classify the malware instance by using n-gram features (profiles) of disassembled malware. In our methodology, we use n-gram feature of the malware to classify the malware instance with respect to their family.

5.3 Our n-gram Based Malware Classifier

As stated in the introduction, current malware samples cannot be analyzed easily based on their statistical features' as in the previous decade because of the increasing use of the obfuscation techniques by the malware authors.

The proposed algorithm consists of preprocessing, training and testing phase. Malware samples are collected through TR-CERT [36] activities in The National Research Institute of Electronics and Cryptology. We classified our dataset by using Microsoft Security Essential (MSE) antivirus tool [37]. In other words, naming of the malware instance is performed by the MSE tool. Malware naming is not a well standardized area where all vendors, players can name and classify malware according to their intentions,

and common sense in naming cannot be achieved among the stakeholders [38]. After that preprocessing step, PEid as a useful tool to inspect PE files, is used to disassemble malware instances [39]. We extract a malware instance's n-gram profile through opcode sequences obtained from PEid. We are using opcode sequences instead of byte sequences of the malware.

In our study, machine codes to extract malwares' n-gram profile instead of byte sequences are considered and the n-gram feature space is considerably reduced. In this manner calculations are performed faster and efficiently. Each malware sample is used to determine its subfamily vector which is named as the centroid of the subfamily.

Family of the malware is a descriptor of the malware used to classify malware samples according to their features especially in terms of the tasks performed and the purpose of the creation. Subfamily is the specialized version of the family that describes malware samples definitely. For instance if a malware labeled as Win32-Ramnit.F by an anti-virus scanner, this means the malware belongs Win32-Ramnit family and Win32-Ramnit.F subfamily.

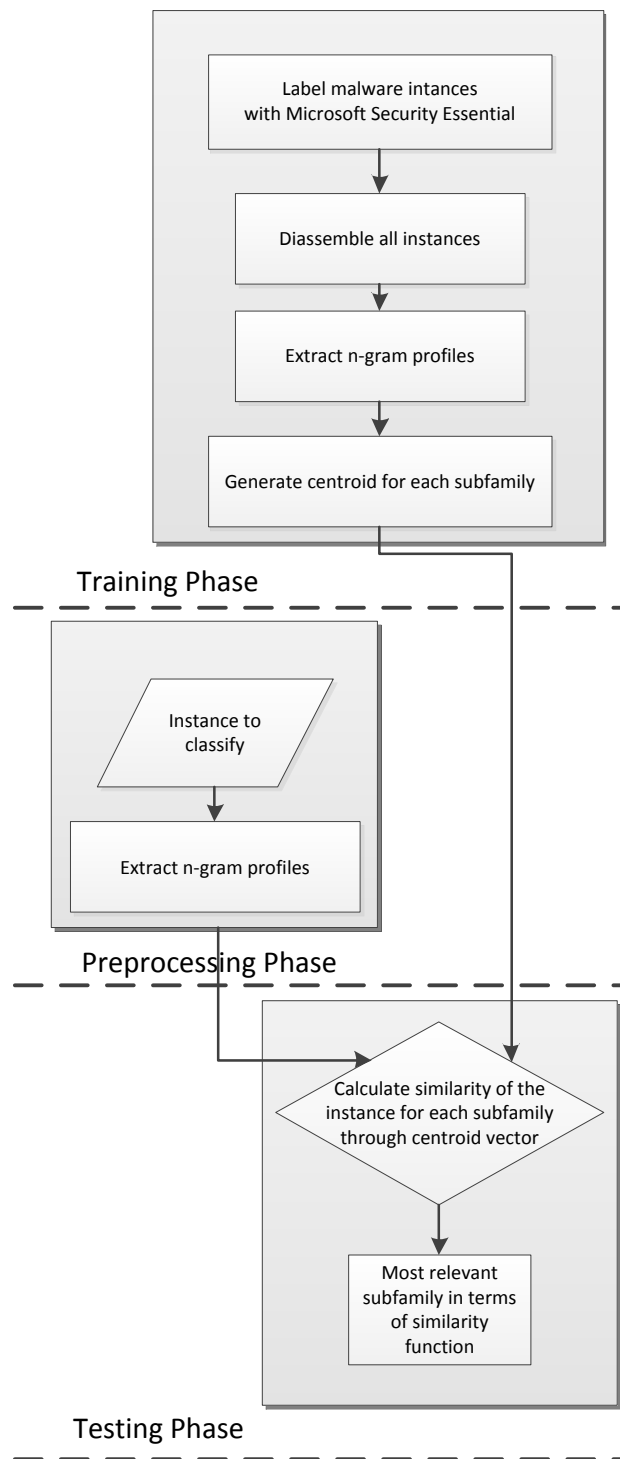


Figure 5.2. Architecture of the Malware Classification System

Centroid of the subfamily comprises the most frequent n-gram of the subfamily instances. In other words, n-grams (words or terms), which occur with higher document frequency in the subfamily instances, are used to construct the centroid vector. So the subfamily is represented by its centroid vector. For instance, centroid of the subfamily is presented by \vec{C}_s as follows:

$$\vec{C}_s = \begin{pmatrix} n\text{-gram with highest } df \text{ value} \\ n\text{-gram with second highest } df \text{ value} \\ \vdots \\ n\text{-gram with } L^{\text{th}} \text{ highest } df \text{ value} \end{pmatrix} \quad (5.1)$$

where df is the document frequency.

To classify an instance, similarity function is calculated by counting the number of matching n-gram (term) for each centroid of the subfamily.

$$Common(C_{s_i}, \vec{m}) = \begin{cases} 1, & (C_{s_i} \in \vec{m}) \\ 0, & \text{otherwise} \end{cases} \quad (5.2)$$

$$Sim(\vec{C}_s, \vec{m}) = \sum_{i=0}^L Common(C_{s_i}, \vec{m}) \quad (5.3)$$

$$Class(\vec{m}) = \max(\cup_{i=0}^{sub\ number} Sim(\vec{C}_s, \vec{m})) \quad (5.4)$$

where m denotes malware whose family is unknown and it will be determined via presented method. \vec{m} is the n-gram feature vector extracted from unknown malware instance denoted by m . Subindice is the subfamily indexing for $s=1, 2, \dots, 15$. The

function, denoted by *Common*, returns 1 if malware n-gram profile (\vec{m}) consists *i-th* n-gram of the centroid of taken subfamily(\vec{C}_s) denoted by C_{s_i} otherwise return 0. Equation 5.3 gives similarity measure between the unknown instance and the subfamily centroid. Similarity measure is the sum of the common n-grams. In Equation 5.4, after all similarity measures are calculated, the unknown instance is classified as the closest centroid's subfamily.

Process flow is illustrated in Figure 5.2. When an instance has two or more equal similarity value for two different subfamilies, an error occurs. However this error will be named as the small error because these two or more equal similarity values for subfamily may belong to the same family. As we know, the subfamilies sustain their common family feature. Other types of error are named as big error.

5.4 Obtained Results

In order to perform our experiments, we collect significantly large malware database as stated in the section 5.3. To obtain more accurate results we count in the subfamilies that contain maximum number of samples in our dataset. In this manner, experiments are carried out 1056 samples belonging to ten families, five of them have two subfamilies, and therefore there exists 15 subfamilies in our dataset. Table 5.1 indicates how many samples were taken from which subfamily in our dataset. This data set consists only a 2% of the original database. The amount of the sample is sufficient to demonstrate whether n-gram centroid of the subfamily may be used to classify malware instance or may not.

Table 5.1 Number of the Instances for Each Subfamily

Subfamily Name	Instance Number	Subfamily Name	Instance Number
Win32-Vobfus.Y	13	Win32-Sality.AT	64
Win32-Alureon.H	19	Win32-Small.AHY	69
Win32-Ramnit.F	19	Win32-Renos.NS	95
Win32-Virut.BG	19	Win32-Sality.AM	100
Win32-Alureon.CT	22	Win32-Renos.LT	137
Win32-Agent.ACF	23	Win32-Vobfus.gen!D	183
Win32-Viking.CR	30	Win32-Ramnit.B	200
Win32-Vobfus.AH	42		

To evaluate our methodology, five-fold cross-validation is used: the selected malwares' subfamilies are randomly partitioned into five disjoint sets of approximately equal size, named as "folds". Training and testing phases are performed five times. At each iteration step, one fold is selected as a testing set, and other four folds are combined to form a training set. Therefore, each sample is used five times for training and once for testing. And the estimated error is computed as the total error generated from the five iterations, divided by the total number of the initial tuples.

There are two main parameters in the experimental setup: the first parameter is the size of the n-grams and the second parameter is the number of the list size which is constituted by ranking the n-grams according to their *df* values in the subfamilies. The size of the n-grams, denoted by *n*, allows us to decide how long in bytes the n-gram will be. In the experiments, tests are run with *n*=3, *n*=4, *n*=5 and *n*=6. The second parameter, denoted by *L*, is chosen to express a subfamily in a simple way. Tests are run with *L*=40, *L*=50 and *L*=60.

Table 5.2 shows the obtained training error over the parameters n and L as well as Table 5.3 shows the resulting testing error. As can be seen from the Table 5.2 and Table 5.3, to increase the size of the n -gram does not produce accurate results every time. Because if the parameter n increases, n -grams cannot capture the subfamily features, in contrary the selected n -grams can only represent a feature specific of the sample. However, the opposite case, namely if the n is chosen very small, n -grams can mostly become the common feature of the all subfamilies as well as all samples.

We achieved the highest success rate when $n=4$ as confirmed by the results in [28] also. Elaborating the parameter choice effects, if the parameter L is increased, the error rate decreases. Since the more common n -gram makes it easy to classify instance appropriately. As maintained in the previous section, the n -gram profiles are captured from the disassembled malware, therefore the space of the n -gram decreases dramatically. For all that, L could not be taken more than 60, due to having very small sized n -gram space (*i.e.*, for Win32-Agent.ACF n -gram feature space is 74).

As a result of the experiment, the most appropriate parameter pair is obtained when $n=4$ and $L=60$. The obtained training and testing errors rate for n and L pairs from our experiment are listed in the following Table 5.2 and Table 5.3, respectively.

Table 5.2. Training Error

N-gram Length	Top L N-gram in the Subfamily Malwares					
	L=40		L=50		L=60	
	Total Error	Without Subfamily Error	Total Error	Without subfamily Error	Total Error	Without subfamily Error
n=3	0.231	0.101	0.150	0.058	0.090	0.024
n=4	0.143	0.056	0.106	0.021	0.053	0.014
n=5	0.124	0.041	0.109	0.024	0.058	0.015
n=6	0.123	0.038	0.115	0.024	0.108	0.019
n=7	0.151	0.031	0.115	0.031	0.098	0.019
n=8	0.125	0.041	0.124	0.037	0.111	0.028

Table 5.3 Testing Error

N-gram Length	Top L N-gram in the Subfamily Malwares					
	L=40		L=50		L=60	
	Total Error	Without Subfamily Error	Total Error	Without subfamily Error	Total Error	Without subfamily Error
n=3	0.262	0.109	0.184	0.066	0.131	0.038
n=4	0.169	0.069	0.141	0.037	0.082	0.023
n=5	0.150	0.056	0.128	0.038	0.082	0.026
n=6	0.143	0.043	0.140	0.027	0.134	0.023
n=7	0.170	0.039	0.140	0.036	0.125	0.025
n=8	0.139	0.042	0.148	0.040	0.138	0.034

6 Fusion

In this section, we will explain the combination of the DMA tool and n-gram based malware classification methods to make cascade malware analyzer tool which is capable of detection and classification of malware. Thus, thanks to the fusion of the two mentioned methods, one file can be easily analyzed.

Figure 6.1 summarizes the performed work and shows the relationship between detection and classification module. We can shortly explain the fusion procedure as follows;

- Execute the executable file which is wanted to dissect with DMA's execute option. Which means the file is executed by pin tool in order to bypass anti-VMware aware techniques.
- If the executable file tries to call the suspicious API which is monitored by our pin tool, the file is marked as anti-VMware aware malware. Besides, the pin tool returns the results of the called API such a native machine, not as a VMware. Thus anti-VMware aware malware continues to run and reveals its behavior.

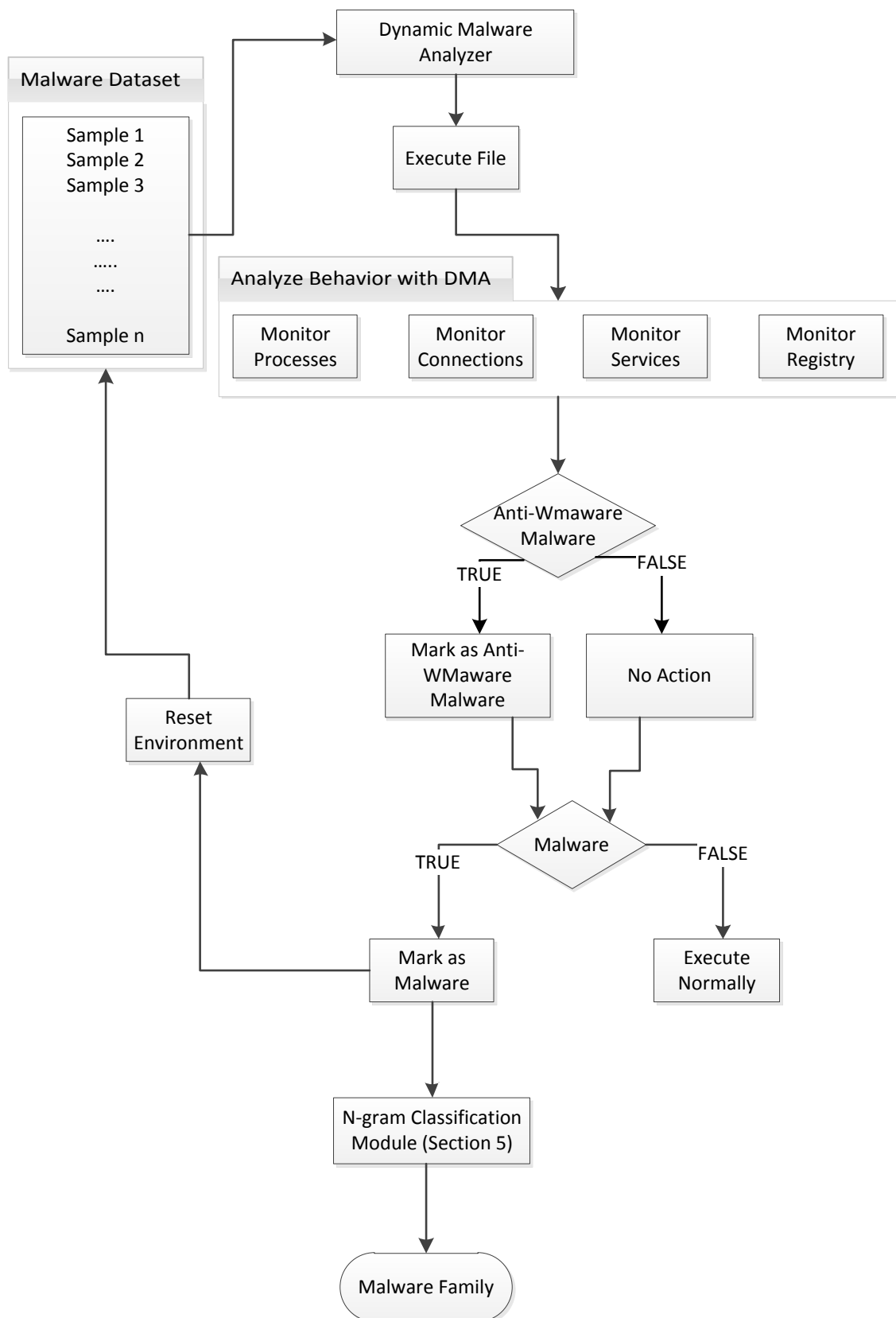


Figure 6.1. Fusion of DMA and n-gram Based Classification Module

- In the meantime, the monitor module of DMA tool track the processes, connections, services and registry changes and if the following situation occurs DMA will alert that the file is malware;
 - The process except that determined process set in 4.3.1 section tries to make connection.
 - A process tries to establish a connection to malicious domain which is determined in 4.3.2 section.
 - A process tries to create a service on the system.
 - A process tries to add key into registry to gain persistence on the system.

- If there occurs any anomaly in the monitored features of the system the file is named as malware. On the other hand if there is not any unexpected situation the file is executed normally.
- After the determining the malicious file we carry out the classification task with our proposed n-gram based malware classification method. Briefly in this method we classify the malicious file by comparing it with the previously trained malware set. As stated before to classify sample it is not executed since the method uses the static n-gram feature of the file. The details are stated in the 5.3 section.

6.1 Evaluation Methodology & Obtained Results

To evaluate the work we have selected only the executable files in the Pahadus public malware set [15]. Test is carried out with the 64 bits Windows 7 operating system and VMware workstation. These malware set is scanned with MSE anti-virus solutions to detect and classify samples. Since the dataset is public, means there exists signature for samples in MSE, all samples detected easily.

Since we have malware detection and classification module in our study we need to evaluate each module separately. So we determined the accuracy of the DMA with the captured malware samples in the testing set. On the other hand, the accuracy of the n-gram based classification module is measured with the ratio of the properly classified sample with respect to MSE to previously detected malware samples.

Before the results, let's give you the details of the dataset. In the dataset there are 72 samples but we didn't execute all the samples. As you can see in Figure 6.2 only 49 (69% of the dataset) samples are executed normally in our test environment.

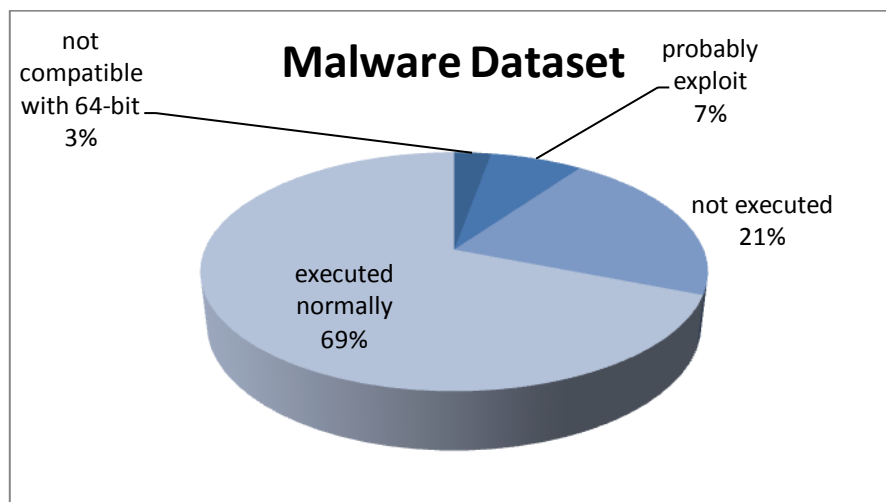


Figure 6.2. Details of Executed Malwares

As seen in Figure 6.3, DMA has detected the 86% (42 samples over 49) of the malware set as a malware. Figure 6.4 shows that which module of the DMA detects the present of the malicious activity. We can obviously say that majority of the malware set is detected by registry and process monitoring features of the DMA. By the way, there are 5 samples that try to create service to gain persistence on the system and 7 samples that try to establish network connection.

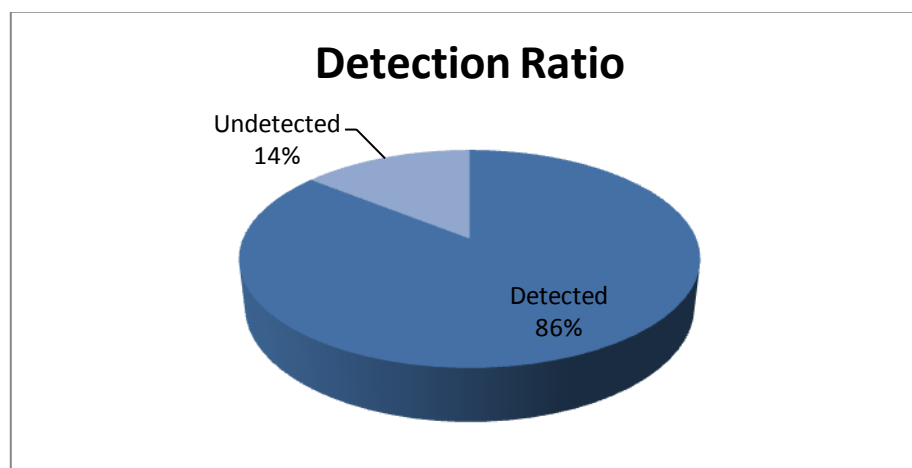


Figure 6.3. Malware Detection Ratio

In the test dataset DMA didn't detect 7 samples. To understand our limitations we wanted to analysis these samples with publicly available dynamic analysis tool like Anubis [40]. While we were performing dynamic analysis with Anubis web service, we realized that undetected malware samples are generally performs file operations such as delete itself, modifies and destruct windows native files, copies itself in the Windows directory to stay undetected by users, etc. By the way the 3 samples were also not detected by Anibus and one of the samples wasn't analyzed because Anibus didn't consider the file as a Windows executable.

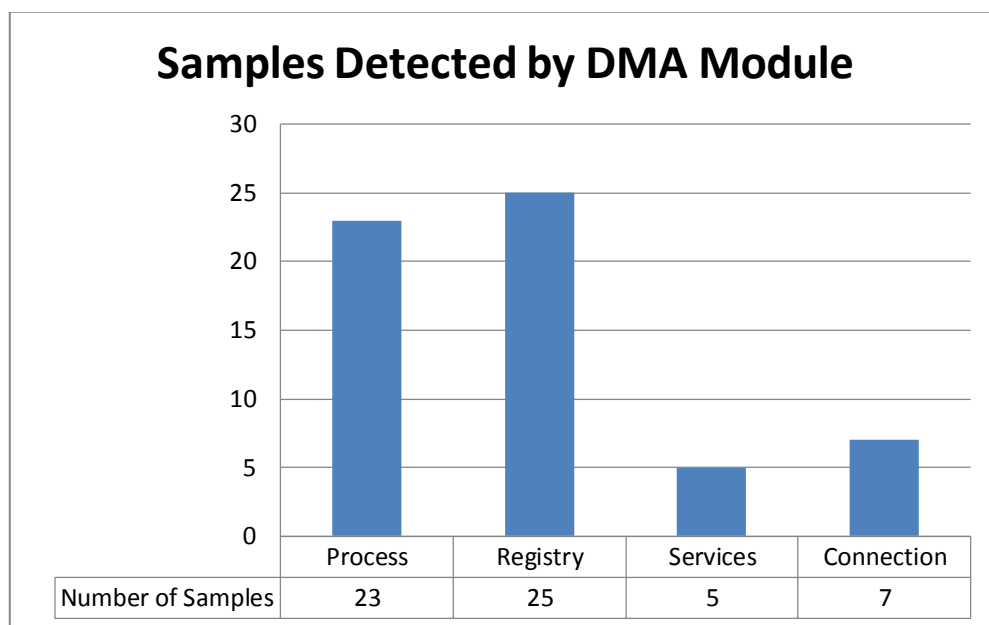


Figure 6.4. Number of Samples Detected by DMA's Modules

As stated before after determining step we classify malware samples with n-gram based classification method. We have taken $L=60$ and $n=4$ which is the best pairs stated in section 5.4. Our n-gram classifier was trained with the same data set in section 5.4. The previously detected 49 malware samples classified %92 accurately with respect to MSE anti-virus solution's report.

7 Conclusion

The main task of this thesis was to identify malwares and then classify them. It is not as simple task as it sounds because of the advanced obfuscation techniques generated by malware authors in order to avoid detection of the anti-malware solutions. In the study we focused on anti-virtual machine evasion techniques to provide secure and reproducible environment to the malware analyst.

We have argued that existing anti-VMware detection methods exists but there is a lack of research to analyze this samples' behavior. Consequently, we have developed our dynamic malware analyzer tool which is called DMA. It can execute anti-Virtual aware malware samples in VMware machine.

Pin [41] is the main trick to bypass anti-virtual machines techniques that is used in our Dynamic Malware Analyzer (DMA). So the first step of the malware detection relies on pin tool. To this end we developed our pin tool based on the study carried out by *Vishnani et al* [12].

The DMA is capable of monitoring system resources such as connections, processes and which are highly used to dynamic analysis methods. It also informs the analyst when Windows Registry is changed.

DMA's detection accuracy is tested over the Pahadus public malware set [15] and the obtained results are pretty encouraging. We have detected malware samples with 86% accuracy. After the detection of malware, the classification carries out with n-gram features of the binary form of the malware. For the classification task we used malware dataset given by BILGEM which is collected with its honeypots. Experimental results show that the classification accuracy for the detected samples when n and L are chosen 4 and 60, is 92% which seems to be very promising.

DMA is currently still under development stage so it can't handle all the tasks automatically. We will add this functionality in the future version of the DMA. On the other hand while detecting the malware samples, there may be scoring mechanism to describe the harmfulness degree of the file. The realization of the scoring mechanism will provide fast, more reliable, less struggle and user-independent analysis.

Besides that, we realized that undetected malware samples by using DMA are generally performs file operations so we decided to add file monitoring feature to the DMA as fast as possible to make it more accurate and functional.

During the course of evaluating DMA with real malware samples, it became apparent that dynamic analysis alone might not be the perfect way to analyze unknown executables.

Finally, to improve the classification accuracy of n-gram based technique, experiments by using large dataset while using variable length n-gram feature vector of the malware is underway.

References

- [1] "Microsoft Security Intelligence Report Volume 12", [Online], Available: "<http://www.microsoft.com/security/sir/>", [Accessed 02.05.2012], (2011).
- [2] "Playstation Network Hacked", [Online], Available: "<http://www.shacknews.com/article/68215/sony-confirms-playstation-network-hacked>", [Accessed 02.05.2012].
- [3] "TIB DDoS Attack by Anonymous", [Online], Available: "<http://www.hurriyet.com.tr/teknoloji/17990950.asp>", [Accessed 02.05.2012].
- [4] "Stuxnet's Target is Iran" [Online],. Available: "<http://en.wikipedia.org/wiki/Stuxnet>", [Accessed 02.05.2012]..
- [5] Sikorski, M., Honig, A., "Practical Malware Analysis", No Starch Press, (2012).
- [6] "ProFTPD Backdoor", [Online], Available: "<http://www.osvdb.org/69562>" , [Accessed 05.02.2012].
- [7] Bacher, P., Holz, T., Kötter, M., Wicherski, G., "Know your enemy: Tracking botnets", [Online], Available: "<http://www.honeynet.org/papers/bots>", [Accessed 02.05.2012].
- [8] "Windows WMI Classes", [Online], Available: "[http://msdn.microsoft.com/en-us/library/windows/desktop/aa394554\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/aa394554(v=vs.85).aspx)", [Accessed 02.05.2012].

- [9] "ScooyNGThe VMware detection tool", [Online], Available: "<http://www.trapkit.de/research/vmm/scooyng/index.html>", [Accessed 19.05.2012]
- [10] "On the Cutting Edge:Thwarting Virtual Machine Detection", [Online], Available: "http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf", [Accessed 19.05.2012].
- [11] Quist, Q., Smith, V., "Detecting the Presence of Virtual Machines Using the Local Data Table", [Online], Available: "<http://tuts4you.com/request.php?2141>", [Accessed 19.05.2012].
- [12] Vishnani, K., Pais, R., A., Mohandas, R., "Detecting & Defeating Split Personality Malware" in *SECURWARE, The Fifth International Conference on Emerging Security Information, Systems and Technologies*, Nice, France, (2011).
- [13] "Joanna Rutkowska's Blog", [Online], Available: "<http://theinvisiblethings.blogspot.com/>", [Accessed 02.05.2012].
- [14] "VMware Backdoor", [Online], Available: "<https://sites.google.com/site/chitchatvmback/backdoor>", [Accessed 02.05.2012].
- [15] "Pahadus Malware Set", [Online], Available: "<https://sim.cert.ee/hw/pahadus.zip>", [Accessed 02.05.2012].
- [16] "Regshot web page", [Online]. Available: "<http://sourceforge.net/projects/regshot/>", [Accessed 19.05.2012]
- [17] "Malware Domain List", [Online], Available: "<http://www.malwaredomainlist.com/update.php>", [Accessed 02.05.2012].
- [18] "Malware Prevention through DNS Redirection", [Online], Available: "<http://www.malware-domains.com/>". [Accessed 02.05.2012].

- [19] M. Egele, T. Scholte, E. Kirda and C. Kruegel, "A Survey on Automated Dynamic Malware Analysis Techniques and Tools", *ACM Computing Surveys*, vol. 44, no. 2, (2012).
- [20] Willems, C., Holz, T., Freiling, F., "Toward Automated Dynamic Malware Analysis Using CWSandbox", *IEEE Security and Privacy archive*, volume 5, Issue 2, pages 32-39, (2007).
- [21] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, G., M., Liang, Z., Newsome, J., Poosankam, P., Saxena, P., "BitBlaze: A New Approach to Computer Security via Binary Analysis", *ICISS '08 Proceedings of the 4th International Conference on Information Systems Security*, pages 1-25, (2008).
- [22] Bayer, U., Kruegel, C., Kirda, E., "TTAnalyze: A Tool for Analyzing Malware", *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, Hamburg, Germany, (2006).
- [23] Garfinkel, T., Rosenblum, M., "A Virtual Machine Introspection Based Architecture for Intrusion Detection", *In Proc. Network and Distributed Systems Security Symposium*, pages 191-206, (2003).
- [24] Lau, B., Svajcer, V., "Measuring virtual machine detection in malware using DSD tracer," *in Proceedings of Virus Bulletin*, (2008).
- [25] Zhu, D., Y., Chin, E., "Detection of VM-Aware Malware", [Accessed 19.05.2012], [Online], Available:
"http://radlab.cs.berkeley.edu/w/upload/3/3d/Detecting_VM_Aware_Malware.pdf"
- [26] Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E., Vigna, G., "Efficient Detection of Split Personalities in Malware, 17th Annual Network", *in 17th Annual Network and Distributed System Security Symposium*, San Diego, USA, (2010).

- [27] Omella, A., A., "Methods for Virtual Machine Detection", [Online], Available: "http://charette.no-ip.com:81/programming/2009-12-30_Virtualization/www.s21sec.com_vmware-eng.pdf", [Accessed 02.05.2012], (2006).
- [28] Kolter, J., Z., Maloof, M., A., "Learning to Detect Malicious Executable in the Wild", *The Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, New York, NY, USA, (2004).
- [29] Abou-Assaleh, T., Cercone, N., Keslji, V., Sweidan, R., "n-gram-based Detection of New Malicious Code," in *The Proceedings of the 28th Annual International Computer Software and Applications Conference*, Washington, DC, USA, (2004).
- [30] Santos, I., Peña, Y., K., Devesa, J., Bringas, P., G., "n-Grams-Based File Signatures For Malware Detection," in *The Proceedings of the 11th International Conference on Enterprise Information Systems*, Volume AIDSS, (2009).
- [31] Burrows, S., Tahaghoghi, S., M., M., "Source code authorship attribution using n-grams," in *In Proceedings of the Twelfth Australasian Document Computing Symposium*, RMIT University, Melbourne, Australia, (2007).
- [32] Frantzeskou, G., Stamatatos, E., Gritzalis, S., Katsikas, S., "Effective identification of source code authors using byte-level information," in *In Proceedings of the Twenty-Eighth International Conference on Software Engineering*, Shanghai, China, (2006).
- [33] Tesauro, G., J., Kephart, O., J., Sorkin, B., G., "Neural networks for computer virus recognition," *IEEE EXPERT Magazine*, pp. 5-6, (1996).
- [34] Arnold, W., Tesauro, G., "Automatically Generated Win32 Heuristic Virus Detection," in *Virus Bulletin Conference*, (2000).

- [35] Schultz, M., G., Eskin, E., Zadok, E., Stolfo, S., J., "Data Mining Methods for Detection of New Malicious Executables," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, (2001).
- [36] "Turkey Computer Emergency Response Team", [Online], Available: "<http://www.bilgiguvenligi.gov.tr/certen/index.php>", [Accessed 19.05.2012].
- [37] "Microsoft Security Essential", [Online], Available: "http://www.microsoft.com/security_essentials", [Accessed 15.05.2012].
- [38] Bailey, M., Oberheide, J., Andersen, J., Mao, Z., M., Jahanian, F., Nazario, J., "Automated Classification and Analysis of Internet Malware," in *Proceedings of the 10th international conference on Recent advances in intrusion detection*, (2007).
- [39] "PEiD tool", [Online], Available: "<http://www.peid.info/>", [Accessed 15.05.2012]
- [40] "Anubis: Analyzing Unknown Binaries", [Online], Available: "<http://anubis.iseclab.org/>", [Accessed 02.05.2012].
- [41] "Pintool", [Online], Available: "<http://www.pintool.org/>", [Accessed 19.05.2012].
- [42] Hart, M., J., "*Windows System Programming*", 4th Edition, Addison-Wesley, (2010)

Biographical Sketch

The author of this thesis was born in 1986 in Balıkesir, Turkey. He has studied in Sırrı Yırcalı Anatolian High School between 2000 and 2004, and started his undergraduate education in the Computer Engineering Department of the Engineering and Technology Faculty of Galatasaray University in 2005-2006 terms. Consequent to the graduation from the undergraduate degree in 2009, he has enrolled to the Computer Engineering Master's Degree in the same university's Institute of Sciences. Since 2009 he has been working at BILGEM as a researcher in Information Security Department.

Article written under the supervision of the Doc. Dr. Tankut Acarman, with the name of "Proposal of n-gram Based Algorithm for Malware Classification" which was presented at SECURWARE 2011 in Nice, France.