Classification des logiciels malveillants basée sur le comportement à l'aide de l'apprentissage automatique en ligne

Abdurrahman Pektaş

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Abdurrahman PEKTAŞ

Thèse dirigée par **Jean Claude Fernandez**
et codirigée par **Tankut Acarman**

préparée au sein **du laboratoire Verimag**
et de **Ecole Doctorale Mathématiques, Sciences et Technologies de l'Information, Informatique**

Behavior based malware classification using online machine learning

Thèse soutenue publiquement le **10th December 2015**,
devant le jury composé de :

Prof. Nicolas Halbwachs
University Grenoble Alpes, France, Président
Prof. Bernard Levrat
University of Angers, France, Rapporteur
Prof. Jean-Yves Marion
Ecole des Mines de Nancy, France, Rapporteur
Prof. Sylvain Hallé
The Université du Québec à Chicoutimi, Canada, Examinateur
Prof. Jean Claude Fernandez
University Grenoble Alpes, France, Directeur de thèse
Prof. Tankut Acarman
Galatasaray University, Turkey, Co-Directeur de thèse

To all my family especially,
to Elif for her endless love and support...

Abstract

Recently, malware (short for malicious software) has greatly evolved and has became a major threat to the home users, enterprises, and even to the governments. Despite the extensive use and availability of various anti-malware tools such as anti-viruses, intrusion detection systems, firewalls etc., malware authors can readily evade these precautions by using obfuscation techniques. To mitigate this problem, malware researchers have proposed various data mining and machine learning approaches for detecting and classifying malware samples according to the their static or dynamic feature set. Although the proposed methods are effective over small sample sets, the scalability of these methods for large data-sets is under investigation and has not been solved yet.

Moreover, it is well-known that the majority of malware is a variant of previously known samples. Consequently, the volume of new variants created far outpaces the current capacity of malware analysis. Thus developing a malware classification to cope with the increasing number of malware is essential for the security community. The key challenge in identifying the family of malware is to achieve a balance between increasing number of samples and classification accuracy. To overcome this limitation, unlike existing classification schemes which apply machine learning algorithms to stored data, (i.e. they are off-line algorithms) we propose a new malware classification system employing online machine learning algorithms that can provide instantaneous update about the new malware sample by following its introduction to the classification scheme.

To achieve our goal, firstly we developed a portable, scalable and transparent malware analysis system called VirMon for dynamic analysis of malware targeting the Windows OS. VirMon collects the behavioral activities of analyzed samples in low kernel level through its developed mini-filter driver. Secondly, we set up a cluster of three machines for our online learning framework module (i.e. Jubatus), which allows to handle large scale data. This configuration allows each analysis machine to perform its tasks and delivers the obtained results to the cluster manager.

Essentially, the proposed framework consists of three major stages. The first stage consists of extracting the behavior of the sample file under scrutiny and observing its interactions with the OS resources. At this stage, the sample file is run in a sandboxed environment. Our framework supports two sandbox environments: VirMon and Cuckoo. During the second stage, we apply feature extraction to the analysis report. The label of each sample is determined by using Virustotal, an online multiple anti-virus scanner framework consisting of 46 engines. Then at the final stage, the malware dataset is partitioned into training and testing sets. The training set is used to obtain a classification model and the testing set is used for evaluation purposes.

To validate the effectiveness and scalability of our method, we have evaluated our method by using 18,000 recent malicious files including viruses, trojans, backdoors, worms, etc., obtained from VirusShare, and our experimental results show that our method performs malware classification with 92% of accuracy.

Keywords: Malware classification, dynamic analysis, online machine learning, behavior modeling

Résumé

Les malwares, autrement dit programmes malicieux ont grandement évolué ces derniers temps et sont devenus une menace majeure pour les utilisateurs grand public, les entreprises et même le gouvernement. Malgré la présence et l'utilisation intensive de divers outils anti-malwares comme les anti-virus, systèmes de détection d'intrusions, pare-feux etc ; les concepteurs de malwares peuvent significativement contourner ses protections en utilisant des techniques d'obfuscation. Afin de limiter ces problèmes, les chercheurs spécialisés dans les malwares ont proposé différentes approches comme l'exploration des données (data mining) ou bien l'apprentissage automatique (machine learning) pour détecter et classifier les échantillons de malwares en fonction de leurs propriétés statiques et dynamiques. De plus les méthodes proposées sont efficaces sur un petit ensemble de malwares, le passage à l'échelle de ses méthodes pour des grands ensembles est toujours en recherche et n'a pas été encore résolu.

Il est évident aussi que la majorité des malwares sont une variante des précédentes versions. Par conséquent, le volume des nouvelles variantes créées dépasse grandement la capacité d'analyse actuelle. C'est pourquoi développer la classification des malwares est essentiel pour lutter contre cette augmentation pour la communauté en sécurité. Le challenge principal dans l'identification des familles de malware est de réussir à trouver un équilibre entre le nombre d'échantillons augmentant et la précision de la classification. Pour surmonter cette limitation, contrairement aux systèmes de classification existants qui appliquent des algorithmes d'apprentissage automatique pour sauvegarder les données ; ce sont des algorithmes hors-lignes ; nous proposons une nouvelle classification de malwares en ligne utilisant des algorithmes d'apprentissage automatique qui peuvent fournir une mise à jour instantanée d'un nouvel échantillon de malwares en suivant son introduction dans le système de classification.

Pour atteindre notre objectif, premièrement nous avons développé une version portable, évolutive et transparente d'analyse de malware appelée VirMon pour analyse

dynamique de malware visant les OS Windows. VirMon collecte le comportement des échantillons analysés au niveau bas du noyau à travers son pilote mini-filtre développé spécifiquement. Deuxièmement, nous avons mis en place un cluster de 3 machines pour notre module d'apprentissage en ligne (Jubatus);qui permet de traiter une quantité importante de données. Cette configuration permet à chaque machine d'exécuter ses tâches et de délivrer les résultats obtenus au gestionnaire du cluster.

Notre outil consiste essentiellement en trois niveaux majeurs. Le premier niveau permet l'extraction des comportements des échantillons surveillés et observe leurs interactions avec les ressources de l'OS. Durant cette étape, le fichier exemple est exécuté dans un environnement *sandbox*. Notre outil supporte deux *sandbox*: VirMon et Cuckoo. Durant le second niveau, nous appliquons des extractions de fonctionnalités aux rapports d'analyses. Le label de chaque échantillon est déterminé à l'aide de Virustotal, un outil regroupant plusieurs anti-virus permettant de scanner en ligne et constitué de 46 moteurs de recherches. Enfin au troisième niveau, la base de données de malware est divisée en ensemble de test et d'apprentissage. L'ensemble d'apprentissage est utilisé pour obtenir un modèle de classification et l'ensemble de test est utilisé pour l'évaluation.

Afin de valider l'efficacité et l'évolutivité de notre méthode, nous l'avons évalué avec une base de 18 000 fichiers malicieux récents incluant des virus, trojans, backdoors, vers etc, obtenue depuis VirusShare. Nos résultats expérimentaux montrent que notre méthode permet la classification de malware avec une précision de 92%.

Mots-clé: Classification de malware, analyse dynamique, l'apprentissage machine en ligne, la modélisation du comportement

Acknowledgments

First of all, I would like to thank my advisors Prof. Dr. Jean-Claude Fernandez and Prof. Dr. Tankut Acarman for believing in me and giving me the chance to work on the malware research. Without their excellent assistance and contributions throughout my research, it would not have been possible to complete this work. Moreover, I'm grateful to Dr. Yliès Falcone for taking care of many administration issues and allowing me to fully focus on my research activities.

Special thanks go to my colleagues at the The Scientific and Technological Research Council of Turkey (TUBITAK), for their invaluable feedback, discussions and collaboration. In particular, I would like to thank Necati Ersen Şişeci, Bakir Emre, Hüseyin Tirli and Osman Pamuk.

Lastly, I would like to thank my wife Elif for her love and support. I would also like thank my parents and my sisters for everything they have done for me and for supporting me throughout my life.

Table of Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

As recent advances in technology have dramatically and irreversibly affected our daily lives over the past few years, we become addicted and chained to the very thing that was supposed to set us free. In particular, as the Internet becomes increasingly ubiquitous around the world, the cyber threats have also become increasingly prevalent and serious. The lack of adequate protection mechanisms on the average users' computer and ignorance and underestimation about security threats have inclined cyber-criminals to launch security attacks.

In the near feature, with IPv6, almost all devices including cars, ovens, baby monitors, TV sets, refrigerators, etc. will have an IP address and will be commanded remotely. Consequently, we will be more prominent targets to malicious actors. Unfortunately, since these electronic devices (known as Internet of Things) have not been designed with enough security in mind, we will face fatal results when these devices get compromised by hackers. For instance one can consider the scenarios where someone hacks a car and then finds a way to inactivate the car's brake system, or if a hacker compromises an oven located in a flat and then fire hazard can be maliciously initiated. Actually, all these scenarios show us how large the field of the cyber space is and how serious outcome can be expected from an attack.

When performing a cyber campaign, the most common and effective way used by attackers is to take advantage of *malware*. Malware, short for malicious software, generally refers to any form of hostile software designed for various purposes such as stealing personal information (e.g. credit card details, user accounts, e-mail lists), using it as a gateway for attacking other hosts, conducting Distributed Denial of Service (DDoS), etc., without the user's consent. For example, worms typically spread through by exploiting server side vulnerabilities over the Internet. Once a target system has been hacked, a malware can install additional payload to control it remotely. In this

way, the victimized system becomes a member of a vast network, called a *botnet* in malware domain. In cyber space, botnets are widely used in launching DDoS attacks, sending phishing emails, hosting vulnerabilities to exploit client-side applications, etc.

The malware community is becoming a commercial industry of cyber-weapons by leveraging the evolving behavior of malware and finding 0-day exploits (exploits for unpatched vulnerabilities) [1, 2]. Current malware has evolved from primitive and replicating viruses that disrupt OS operations and destroy user files to highly evasive and flexible pieces of software that allow cyber-criminals to launch ever-increasingly sophisticated and targeted attacks. Practically, the malware community targets money, corporate espionage and ideological purposes. Even some states/regions develop malware in order to enforce their political, diplomatic, and military tactics. For example, Stuxnet, discovered in 2010, is one of the most known malware targeting Iran's nuclear facilities. Stuxnet has many advanced features such as 4 0-day exploits, user-mode and kernel-mode rootkit capability under Windows OS, a digitally signed driver by two certificate authority [3]. Its creation cost is estimated to be US$1 million [4].

Unfortunately, even though malware production and complexity dramatically increased, the knowledge required by an attacker to deploy malware has considerably decreased over the last decades. This relation between the threat and money investment clearly illustrates the destructive impact of research on malware development targeting large scale of computer systems. This situation stems from the growing usage of user-friendly and automated malware creation frameworks, such as Metasploit or exploit kits. With these attack kits, cyber criminals can easily and automatically launch attacks that infect computers in order to perform their own malicious aims. Some of these tool kits are free, open source, and others are sold on the Internet's black market.

Additionally, as the process of creating a malware sample from scratch is a highly complex and tedious task and requires considerable skills and effort, malware authors employ runtime packers and obfuscation mechanisms [5, 6, 7] (polymorphism and metamorphism). This leads to the explosive increase in malware variants, which are behaviorally identical but statically different samples. According to Cisco, around 400,000 malware variants have been detected every day during 2014 [8]. It is well known that the majority of these malware is the variant of the previously known samples. On the other hand, identifying the family of a malware spares the re-analysis of sample instances and enables researchers to focus on new or unseen malware instances. Therefore, the classification of malware samples into appropriate families is as important as malware detection.

From Figure 1.1, we can observe that the number of malware samples per day has

nearly doubled year-to-year basis, and the number of samples appeared in 2014 has reached at 900,000. In fact, the total of malware produced in 2014 alone is more than the sum of all malware created over the last decade. Unfortunately, this dominant trend is likely to strengthen in the future, and malware will remain the greatest security threat to computer users.

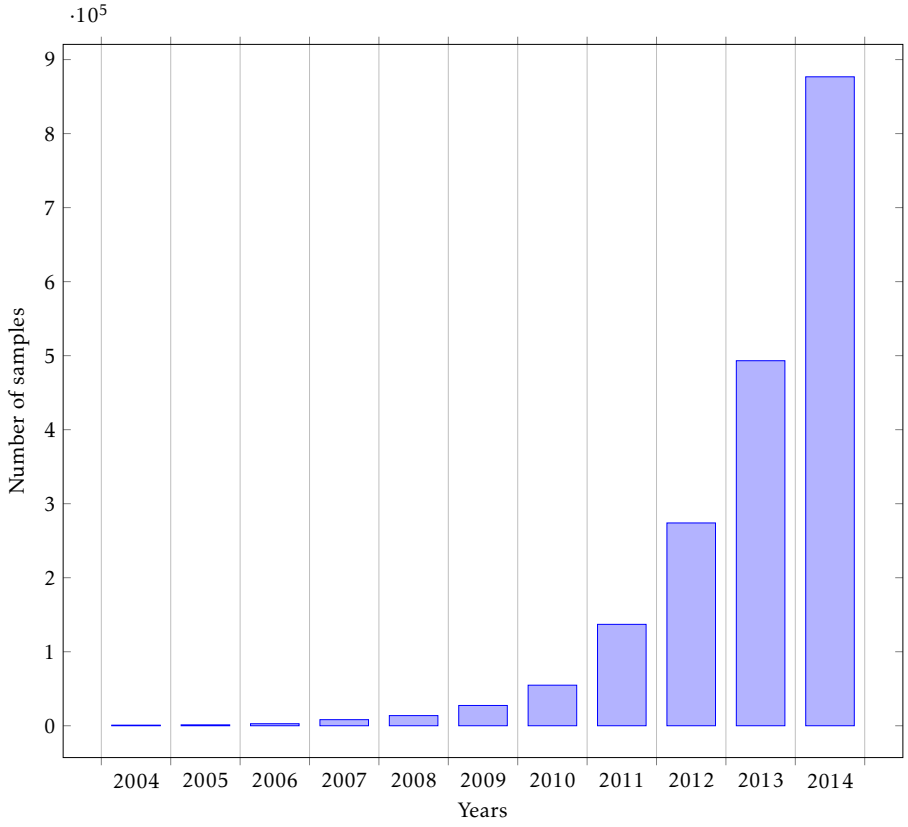


Figure 1.1: Number of malware samples per day [9]

Currently, state-of-the-art malware detection relies on static malware detection which consists of analyzing files without executing them and generally using a signature database or rule-set to operate. Although these solutions are very effective and fast for known malware, they often have low detection rates on unknown instances and variants of previously known samples. Thus, researchers have proposed dynamic analysis methods which are more robust to obfuscation techniques. They model run-time behavior according to the actions that operate on the security-critical OS resources.

First and foremost, the crucial stage of dynamic analysis is to obtain the runtime behavior of a given file appropriately. Since cyber criminals are continuously armoring their malware with anti-analysis techniques such as anti-debugging, VM-detection, logic bombs, etc. to make them more resilient, withstanding with detection, appropriate countermeasures must be considered by researchers. Moreover, the most important thing to note here is that, when detecting malware, often a single solution is not suffi-

cient. Indeed, using advanced multi-layer approaches to identify anomalies caused by malware is essential to obtain a secured network. Even if one of the security solutions is bypassed, other solutions will defend the network against threats.

While automated dynamic analysis has shed light on the activities of programs once they are executed in a controlled environment, there must be some mechanism to determine whether the file is malicious or not. In the literature, researchers generally employ scoring mechanisms and behavioral patterns to assess harmfulness. However, there is not enough research in the field of the automatic classification of malware and similarity of malware families based on their behavior. On the one hand, the proposed methods suffer from the *curse of dimensionality* which limits the feature space to avoid explosion of searching complexity level. On the other hand, while malware features grow proportionally with the number of activities of a given sample, the algorithms require longer execution time and make the analysis inefficient.

1.1 Problem Statement

The number of malware has dramatically increased through the greater use of obfuscation techniques over the last decade. (Figure 1.1 shows this exponential growth.) With these techniques, even the most popular anti-virus products can easily be evaded. To cope with that problem and to provide more automation in the arms race between malware authors and analysts, there is a strong need for new and scalable malware analysis environments to automatically analyze large numbers of malware samples in real-time.

While automated dynamic analysis exploits the activities of a file during its execution in a controlled environment, a mechanism is still required to determine the malicious impact of a file. Automatically classifying malware and similarity of malware families based on their behavior is a technical challenge towards an accurate and reliable analysis and classification. The usual issue with the above method is that, when the feature space increases, data become sparse and the computation time of algorithms increases exponentially with the number of malware samples. (This problem is known as the “curse of dimensionality"). However, unlike existing malware classification techniques, the proposed distributed machine learning method is not only efficient and computationally less expensive but also more adaptable to the model changes at runtime.

This thesis is motivated by the need to classify malware samples in large scale, and aims to propose an optimal classification method.

1.2 Contributions

The contributions of this thesis are as follows:

1. We present Virus Monitor (VirMon), a portable, scalable and transparent system for dynamic analysis of malware targeting Windows OS. VirMon is deployed as an automated and virtualized platform. VirMon collects the behavioral activities of analyzed samples in low kernel level. VirMon is capable of using all recent versions of Windows as an underlying analysis environment while it can successfully make analysis of malware targeting the latest version of Windows. The features of VirMon are as follows:

   (a) VirMon is a *fully-automated* dynamic analysis system: it performs analysis without any human intervention.

   (b) VirMon is *scalable*: the analysis capacity, i.e., the average number of analysis per minute, can be increased by connecting new virtualization servers to the existing system. VirMon's capacity can be improved by adding new analysis machines upon the increase of the analysis workload.

   (c) VirMon supports Network virtualization: the network traffic of analysis machines is distributed to different network locations via VPN to masquerade their IP addresses. This decentralized design approach ensures that the analysis system is not detectable by malware's network level precautions such as comparing public IP addresses of the analysis system.

2. We propose a new malware classification method based on behavioral features. File system, network, registry activities observed during the execution traces of the malware samples are used to represent behavior-based features. Existing classification schemes apply machine-learning algorithms to the stored data, i.e., they are off-line. We use on-line machine learning algorithms that can provide instantaneous update about the new malware sample by following its introduction to the classification scheme. In particular, the proposed malware classification system makes the following contributions:

   (a) It represents malware with its behavioral profile obtained from dynamic analysis.

   (b) It can classify malware samples based on their behavioral profile.

   (c) The proposed system is not only efficient and computationally less expensive but also more adaptable for model changes at runtime.

(d) The samples used for evaluation are shared on [10] for research purposes with other researchers.

1.3 Dissertation Overview

Including this chapter, this dissertation contains a total of seven chapters. An overview of each chapter is provided below:

Chapter 2: Background

Chapter 2 provides a broad overview of the field of malware research. It begins with the definition of malware, and then describes the common types of malware and their key features. We present the history of malware in Section 2.2. In Section 2.3, we provide an in-depth explanation of malware infection methods. Section 2.4 describes the current hacker utilities to deploy cyber attack. In Section 2.5 the exiting anti-malware techniques to evade detection attempts are presented. Finally, we report on the state-of-the art in malware analysis methods by describing the pros and cons of each method.

Chapter 3: Related Work

Chapter 3 discusses the state of the art of malware detection and classification methods with special emphasis on data mining based method. We also highlight the malware representation methods used in that study as well as classification algorithms and dataset. Furthermore, well-known dynamic analysis frameworks are discussed and compared with our proposed framework.

Chapter 4: VirMon: A Virtualization-Based Automated Dynamic Malware AnalysisSystem

Chapter 4 gives the implementation details of the proposed automated dynamic analysis framework named VirMon. First, the chapter explains VirMon's network virtualization infrastructure whose main aim is to mask the IP addresses of the analysis machines. We then elaborate on VirMon's dynamic analysis system components and their features including the collection method of behavioral activities through kernel-callbacks and network activities through designed network components. Following that, we describe the procedure followed by VirMon to analyze a submitted file. Fi-

nally, we report the functional testing results about a mini-filter driver that we developed both on Windows 7 and on Windows 10 (the newest version of Windows).

Chapter 5: Runtime-Behavior based Malware Classification Using Online Machine Learning

Chapter 5 details the approach used to classify malware samples according to the behavioral artifacts in dynamic analysis framework. After a general overview of the proposed method, we provide the methodology employed for modeling a malware sample based on runtime features. Especially, we present the system improvements that we have done in order to analyze a large amount of samples in a short time. Then, we present online learning and its main advantages leveraged in this thesis. Furthermore, we elaborate on state-of-the art online machine learning frameworks and motivate our choice of the Jubatus framework. Then we describe distributed machine learning environment used in this work. Finally, we explain the online machine learning algorithms employed in our study.

Chapter 6: Evaluation

Chapter 6 evaluates a dynamic feature-based malware classification system, for efficiently classifying a large number of malware programs into their respective families. First, we describe the malware dataset that we used in detail. Then, we define the evaluation metrics. Finally, we present and evaluate the obtained results. After describing our approach, we describe an implementation and subsequently an evaluation using malware samples and common applications.

Chapter 7: Conclusion & Future Work

Chapter 7 provides a summary of the contributions and objectives achieved in this research. Moreover it presents some avenues for future work.

Chapter 2

Background

This chapter defines basic terms and definitions in malware research and describes common types of malware in the wild. Section 2.2 gives the history of malware evolution with special emphasis on game challengers type malware. In Section 2.3 malware infection methods are presented. Then we present anti-malware techniques used to evade malware analysis. Finally we present the state of the art in malware analysis.

The term malware, short for malicious software, refers to a piece of software code that works on any computer system for the attacker without knowledge of the system owners. Malware has great popularity among cyber criminals since it offers attractive income opportunities. This popularity makes malware an important threat for the computing society. In this chapter, as malware research is an interdisciplinary and complex research field, we define some important terms and concepts for the sake of clarity.

2.1 Common Types of Malware

There are various approaches to classify malware into certain categories according to given characteristics such as propagation, infection, stealth, exfiltration, command and control (C&C) or concealment techniques, the set of behavior exhibited during run time on the operating system (OS). Furthermore, it is becoming increasingly difficult to identify malware types since nowadays malware authors can easily reach the source code of several malware samples and combine their functionalities to create new and compact ones. Moreover, it is becoming increasingly popular for malware samples to

have an update mechanism for extending their capabilities. For example, one sample can extfiltrate user's credit card information and credentials meanwhile adding a plugin in order to gain system level authority on OS. Interested readers can look at an example to impersonate user tokens after successful exploitation with *meterpreter* agent [11].

Even though there is no general consensus on malware taxonomy, the common malware types and their purposes can be briefly described as follows.

2.1.1 Virus

A virus is a program that needs another program to activate itself. It can replicate itself but generally does not pursue any goal related to network activities, such as infect another host, exfiltrate information from infected machine, etc. Some viruses are written to corrupt the OS or make very harmful activities on the OS while others are harmless and written for personal reputation. As viruses are the oldest malicious program, nowadays most people use the virus term to indicate any type of malware. Because of the common usage of viruses among people, popular security companies prefer to name their products with virus term, e.g. *Virustotal*.

2.1.2 Worm

A worm is a software that runs autonomously; no host program is needed to launch it (e.g. without direct human interaction). A worm has the ability to reproduce itself via computer networks by:

- exploiting vulnerabilities builtin OS services as well as third-party network services,
- social engineering; tricking users into performing the willing actions such as filling and sending user credentials in fake website, opening malicious attached file, etc.,
- leveraging misconfiguration of the network applications (web server, file sharing),
- brute force attack with default username and password pairs.

Since the propagation of worms happens silently in the background, the victim is typically not aware of the infection. In most cases, worms have malicious payloads executed just after the infection phase. Conficker, also known as Kido, is one of the

most famous worm in the computer history and targeting Microsoft Windows OS. The very first sample of Conficker was detected in October 2008 and infected millions of computers all around the world. Conficker exploits a vulnerability of network services located built-in Windows OS including all versions from Windows 2000 to Windows Server 2008 OSs to propagate through the Internet. This vulnerability, named **MS08_067** [12], allows an attacker to execute remote code, thus taking control of the computer remotely.

2.1.3 Trojan

A trojan is any program that seems very useful for users and encourages them to install. Indeed, however, this program also contains hidden malicious payload which may take effect after execution and can lead to many undesirable results. Since the program works correctly, it is very difficult for an ordinary computer user to figure out the effects of the program. Today's trojans have very advanced features from taking complete control of OS including all processes to capturing all key strokes.

For instance, Poison Ivy [13] is a well known trojan that gives the attacker full control the infected user's computer. Poison Ivy was first detected in 2005 and is also known as **remote access trojan (RAT)**. As Poison Ivy RAT can be found easily in the Internet, it was used in many cyber attacks including the RSA SecurID data breach in 2011 [13, 14] and stealing secrets from the chemical industry in USA [15]. Once trojan is activated, it can perform key logging, capture screen shot, record video through camera, sniffing network for critical information, and so on.

2.1.4 Backdoor

A backdoor is an application allowing an attacker to connect to the computer, bypassing the security mechanism of the system with some secret methods hidden in the software. Backdoors provide the attacker with a remote shell (cmd.exe, bash, or special console) on the system to control system remotely. Hackers commonly use backdoors to hide their existence on the system after compromising the system. Besides that, technical support teams sometimes use backdoors for providing help to computer users.

c99.php [16] is one of the most popular PHP backdoor used for web-based attacks [17, 18]. Once c99.php file is uploaded on the system, the attacker obtains complete access on database and sensitive directory which is accessible by the user of web server, e.g. apache user. Moreover, c99.php comes up with some default commands, like privilege escalation to gain root access on the system.

2.1.5 Adware

An adware or advertising-supported software is any software which records user's information such as visited websites, purchased products from web, web search queries, etc., for advertisement purposes. Generally, adwares are integrated into the legitimate software by developers to recover their development costs by selling user's shopping habits to the related companies or showing commercial advertisements. Usually adwares change the preferred home page and search engine to different sites that make money. If the user knows and confirms this process, this type of software can not be named as malware by definition.

In February 2015, at the time of writing this thesis, a pretty shocking adware sample came to light. Lenovo, one of the biggest computer manufacturer has shipped some of the computer with pre-installed adware that compromises all secure connections (SSL traffic) through its trusted root certificate [19, 20]. Even more interestingly, this certificate employs a deprecated version of SHA1 that can easily be cracked by hackers (interested readers can refer to [21]). The pre-installed adware, called Superfish is also capable of hijacking legitimate connections, inject advertising in web pages and monitoring user activity.

2.1.6 Botnet

A botnet is a term used to refer a software that remotely controls collection of compromised computers on behalf of attacker. These computers connect to command & control (C&C) servers by different means of communication protocols such as IRC, HTTP/HTTPS or peer-to-peer and then wait for commands to execute. Botnets are generally used to carry out distributed denial-of-service (DDoS) attacks. Moreover, attackers can take advantage of botnet to steal private data from infected machine, to send spam e-mail, to launch additional attacks, etc.

Zeus/Zbot [22, 23] is probably the most famous botnet ever discovered and has been employed from 2007 to today in many financial cyber attacks. Zeus mainly targets banking information of the compromised computer such as credit card numbers and bank account credentials. In 2011, after the source code of the Zeus was leaked on the Internet, the number of Zeus variants has dramatically exploded.

2.1.7 Spyware

A spyware is computer software that covertly collects personal information without user's informed consent. A spyware generally has the ability to log keystrokes, record

web history, enumerate username and passwords, scan sensitive documents on the hard disk, etc.

Flame [24, 25], also known as Skywiper, is one of the most complicated spyware first discovered in May 2012 and designed to perform espionage activities in Middle East counties including Iran, Syria, Lebanon, etc. Flame consists of multiple complex plugins which enable itself to steal sensitive information and is conjectured that it was not detected for 5 years. Moreover, Flame has worm capabilities which allows it to infect other computers on the network by exploiting network services.

2.1.8 Rootkits

A rootkit is a software designed to hide the existence of certain files, network connections, or processes from computer user by modifying OS settings, mechanisms or structures in order to avoid detection and stay concealed. For example, a rootkit can prevent a process from being visible by the *tasklist* tool in Windows. Conventionally, rootkits come up with backdoor functionality to access the infected computer remotely.

In 2005, Sony BMG Music Entertainment company used a software, named XCP (Extended Copy Protection), to enable copy protection on CDs. Soon after the release of XCP Mark Russinovich published an article describing his first findings about XCP in his blog [26]. Russinovich disclosed that XCP has a rootkit component to hide its existence in the system and also noted that the End User License Agreement(EULA) does not mention this feature. Even worse the software leads exploitable security vulnerability and does not support uninstallation. Following that, Sony quickly released an uninstaller to remove the rootkit component of XCP. However, this did not prevent Sony from paying thousands of dollars in penalties.

2.1.9 Ransomware

A ransomware(or simply ransom) is a software used to restrain the user from accessing computer resources and demands a ransom to release restriction. Some of the ransom encrypt important files on the system while others change password or lock the computer system.

CryptoLocker [27, 28] is one of the famous ransom type malware targeting Windows OS all around the world. CryptoLocker spreads by tricking users into executing a file attachment in fishing e-mail. Once the user opens the attached file, CryptoLocker activates itself and encrypt sensitive documents located on all hard drives of the system by using the RSA encryption algorithm. After encryption, CryptoLocker translates

its private key to the attacker to request a ransom from users of the infected system.

2.2 A Quick Look at the History of Malware

Since the creation of the first malware, security researchers have been continuously combating with malware. In this section we will highlight the history and evolution of malware and specify the most important ones.

In the literature, history of malware is split into several periods [29, 30]. However, generally, the malware evolution can be considered in three phases as shown in Figure 2.1. In the first age, malware was created for proof-of-concept. Then in the middle age, as the Internet become popular, malware is developed for fun and personal reputation. Finally, in the new age, malware was made mainly for financial gain, espionage and sabotage.

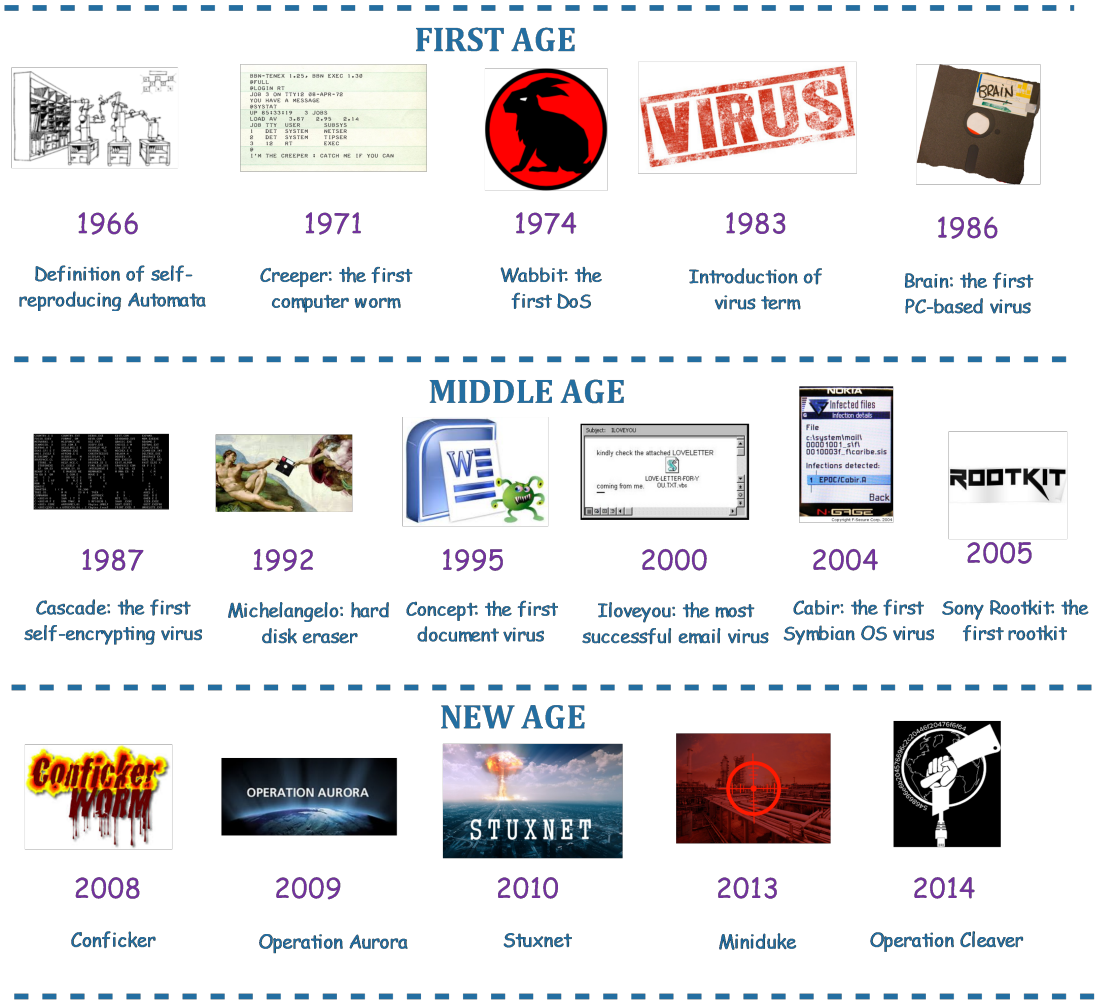


Figure 2.1: Evolution of malware

1966 - Definition of self-reproducing automata

John von Neumann, has explored and gave lectures about the theory and organization of complicated automata since 1949. In 1966, he published a paper called "Theory of self-reproducing automata" [31] pointing out that a computer program could reproduce itself.

1971 - "Creeper": the first computer worm

Bob Thomas wrote a proof of concept program that self-replicate it-self and spreads across the network called the ARPANET, the Internet's ancestor. Once it infects the system, it displays the message "I'm the creeper, catch me if you can!".

1974 - "Wabbit" : the first DoS software

The Wabbit makes incessant copies of itself at high rate that causes the computer to crash because of the overloading.

1983 - Introduction of the term *virus*

While carrying out his dissertation, Frederick Cohen, computer scientist, defines the term **virus** as: "We define a computer 'virus' as a program that can 'infect' other programs by modifying them to include a possibly evolved copy of itself." [32, 33].

1986 - "Brain" - the fist PC-based virus

The first virus for MS-DOS, Brain, was written by two brothers in Pakistan. Brain is a harmless boot sector virus, infecting the first sector of floppies as they are inserted into an infected computer. Interestingly, two brothers inserted their company name (named as 1986 Brain & Amjads (pvt) Ltd.), address and phone number into Brain's source-code.

1987 - "Cascade" - the first self-encrypting virus

Cascade is the first self-encyting virus, infecting COM files and displaying text falling down on the screen. Cascade led IBM to develop its own anti-virus product.

1992 - "Michelangelo" - Hard disk eraser

The Michelangelo virus was designed to run on March 6th; the Italian Renaissance artist Michelangelo's birthday. Once it activates, it erases a computer's hard disk and causes it to crash.

1995 - "Concept" - The first document virus

Concept is the first document or macro virus targeting Microsoft Word. Concept

spreads by exploiting the macros in documents.

2000 - "Iloveyou" - the most successful e-mail virus

The virus was distributed in an email with "I love you" subject containing a malicious attached file. If this attachment is opened, it automatically infects the computer and transfers itself to the user's contacts. Then, this virus downloads and executes additional software from the Internet.

2004 - "Cabir" the first Symbian OS virus

Cabir is the first mobile phone malware targeting Symbian OS. Cabir spreads via a Bluetooth connection and displays the message "Caribe" on the phone's screen. Cabir does not perform any malicious activity on the device but causes the battery of the phone quickly run out because it activates the Bluetooth module to spread to other cell phones.

2005 - "Sony Rootkit" the first rootkit

Sony BMG developed a software to protect copyright of their CDs. Basically, this software hides some files so that users could not duplicate them. This software is not only illegal but also potentially harmful because it contains vulnerabilities allowing attackers to infect the computer.

2008 - Conficker

Conficker is a computer worm targeting the Microsoft Windows OS that first started spreading in November 2008. Conficker uses security flaws in the SMB service allowing arbitrary remote code execution and brute force technique cracking weak passwords of admin account to propagate. The Conficker worm infected millions of personal computers, as well as government and business computers all over the world.

2009 - Operation Aurora

Operation Aurora is a highly sophisticated attack targeting dozens of IT giants such as Google, Yahoo, Juniper Networks, Adobe Systems. It employs a drive-by-download attack to infect computers. To this end, malware authors uses a 0-day (unknown) Internet Explorer vulnerability as an entry point into the systems.

2013 - Miniduke

The Miniduke campaign is an advanced persistent threat, that uses a zero-day exploit located in Adobe Reader to infect targeted government and corporate organizations in Europe. The malware empolyed in the Miniduke campaign focuses on espionage and data-stealing activities and has better defenses against security tools, e.g. anti-viruses.

2014 - Operation Cleaver
Operation Cleaver is a hacking campaign targeting critical industries and organizations such as military, oil and gas, airlines, hospitals, etc. around the world. Operation Cleaver is generally believed to be planned and executed by Iran for retaliation to Stuxnet, Duqu and Flame.

2.3 Malware Infection Methods

In this section, we look at infection methods used by malware authors to invade systems. These methods are constantly evolving by adapting new techniques, finding new weaknesses of users and computer systems, and avoiding defense mechanism. The infection methods can be mainly split into the flowing groups:

- Exploiting Vulnerabilities
- Social Engineering
- Configuration Issues

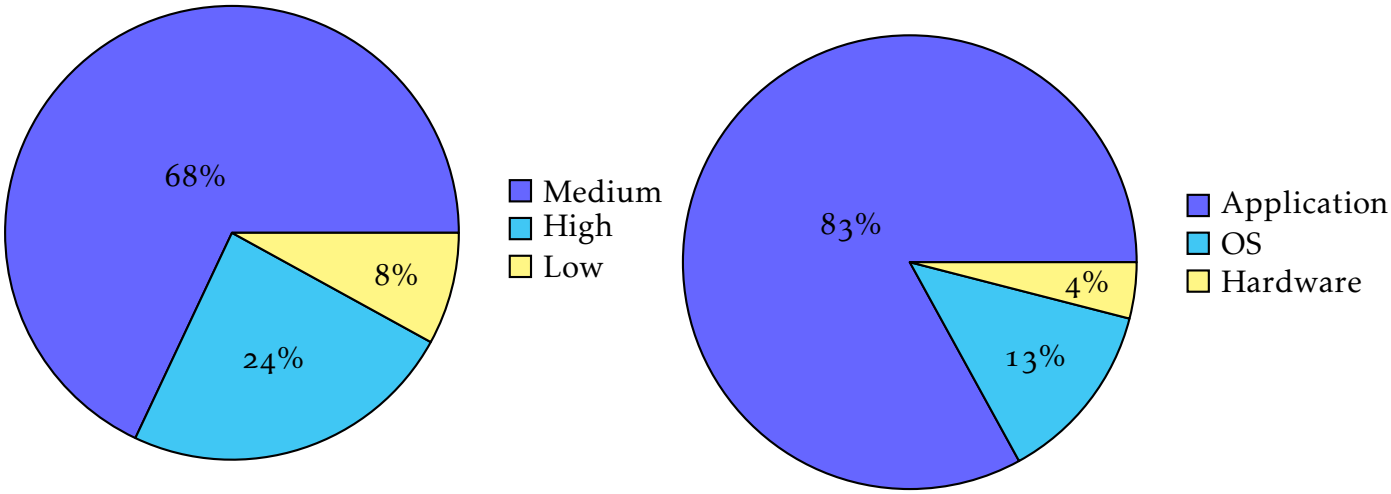
2.3.1 Exploiting Vulnerabilities

Generally, today's malware authors equip their malware with exploit components to take advantage of the weaknesses of an application (e.g. bugs). These applications can be split into two categories:

Client-side: Common end-user software like web browsers, PDF readers, document editors, run-time environments which do not bind a port to listen incoming requests are named client-side applications. The attacker creates a file to exploit vulnerabilities of the client-application. Once the user opens the file, his computer covertly infects.

Server-side: Server applications bind to a port to be accessed remotely. In this case, there is no need to interact with the user's computer. This malware directly accesses and exploits vulnerabilities. These types of vulnerabilities can be found in the implementation of web, DNS, database servers and so on.

Drive-by-download is an attack targeting vulnerabilities of web browsers or web application plugins which refers to the unintentional download of malware into a computing system [34, 35, 36]. To infect through a drive-by-download attack, it is enough to visit a web page. After visiting a malicious web page, the computer downloads and



(a) By severity (b) By product type

Figure 2.2: Vulnerability distribution in 2014 [37].

executes a software without noticing. The initial downloaded software is generally small whose job is to download additional content or executable into computer. This software is named downloader or dropper.

In 2010, a highly sophisticated attack, named as **Operation Aurora** [38, 39], which aimed at stealing intellectual properties of dozens of the giant company such as Google, Yahoo, Juniper Networks, Adobe Systems, employed drive-by-download attack vector to infect computers. To this end, malware authors use a 0-day (unknown) Internet Explorer vulnerability as an entry point into systems. Once the victim visited the previously poisoned web pages of the known web site (known as a watering hole attack), his computer gets infected without notice.

Currently, hackers try to find 0-day vulnerabilities ("0-day exploits", or just "0-days"), exploitable vulnerabilities that the software developer is not aware of and for which no patch is available for the time being. Zero-days are more desirable than any other vulnerabilities for hackers since everyone is vulnerable. Consequently, 0-days are generally used for highly advanced targeted attacks and sold on the black market with good price. Table 2.1 shows a price list for some 0-days on the black market.

For instance, in April 2014, a security flaw was found in the OpenSSL library which is actively used in any encrypted application such as mail servers, VPN end-point, web servers. This security vulnerability allows attackers to steal any information located into application's memory; passwords, private-keys, mail attachments and much more. Moreover, the heart-bleed vulnerability affects client applications. [41]

Indeed, the main reason for the OpenSSL vulnerability is trusting user input. An attacker can trick OpenSSL into allocating a 64KB buffer, copy more bytes than is necessary into the buffer, send that buffer back to the attacker. Thus in each request,

Table 2.1: Prices of Some 0-day Vulnerabilities, [40]

Target Software or OS	Price	Year
Vista	$50,000	2007
Adobe Reader	$5,000 - $30,000	2012
Android	$30,000 - $60,000	2012
Chrome or Internet Explorer	$80,000 - $200,000	2012
Firefox or Safari	$60,000 - $150,000	2012
Flash or Java Browser Plug-ins	$40,000 - $100,000	2012
iOS	$100,000 - $250,000	2012
Mac OSX	$20,000 - $50,000	2012
Microsoft Word	$50,000 - $100,000	2012
Windows	$60,000 - $120,000	2012

the attacker leaks 64 KB of content from the application's memory. This vulnerability has existed in the wild since OpenSSL version 1.0.1, which was released in March 2012. [42]

2.3.2 Social Engineering

The attackers do not always use advanced hacking methods or tools to hack into computer. They also use human factor to bypass security measures to achieve their bad intentions. In the context of information security, this type of attack is called *social engineering*. More formally, social engineering is the art of manipulating people into performing some actions that the attacker wants. Social engineering attacks can be divided into two main categories:

Human-based: In human-based scenarios, the attacker directly interacts with humans, such as calling him as an important person or technical support to retrieve information or download and execute some software to user.

Computer-based: Contrary to human-based scenarios, in computer-based scenarios, attackers do not directly interact the victim. Instead they use digital methods to achieve their goals. For example, sending a phishing e-mail to harvest user credentials, sending malicious attachments in order to get access into victim computer.

Sometimes, social engineering attacks may be very diverse and complicated to understand. Kevin D. Mitnick, one of the most famous hackers, shares some of his hacks by using social engineering methods [43]. Interested readers can also refer to [44, 45].

To attack restricted networks, cyber criminals generally employ removable drives, especially USB sticks. Cyber criminals add malicious content into USB sticks. This malware is directly activated by the auto-run feature provided by Microsoft Windows. On the other hand, even if the user disabled auto-run functionality, an attacker might use removable media with related exploits to hack into the computer. For example Stuxnet [3] uses such kind of 0-day exploit [46] to install itself into a victim's computer.

2.3.3 Misconfiguration

When an application does not include any security bugs, it does not mean it is not exploitable. Many applications come with a bunch of options and when not configured appropriately, these options provide a mean for malware to take control of the systems. Besides that, a malware can log into a system with default installation user names and passwords. All of these misconfigurations allow the attacker unauthorized access to sensitive information.

For example, if a Tomcat manager application is installed with default settings, an attacker can upload a Web application ARchive (WAR)which contains malicious payload that allows the attacker to gain access to the system [47, 48].

2.4 Hacker Utilities

In the past, cybercriminals created malware from scratch but today they utilize highly skilled attack kits; also known as *crimeware*. With attack kits, cybercriminals can easily and automatically launch attacks that infect victim's computer in order to perform their own malicious aims. Some of these toolkits are free, open source, and others are sold in the Internet. In this section, we briefly mention current hacker utilities heavily used to compromise targeted users, computers or networks.

2.4.1 Exploit Kits

An exploit kit, or exploit pack, is a toolkit used by attackers to automate the malware infection process by exploiting client-side applications, especially web browsers and their plugins (e.g. Flash, Java, and PDF). These tool kits play an important role in malware distribution and generally are traded in the online black market- website or forum to advertise malicious software & service, stolen data, etc. As exploit kits are easy to setup and use, there is no need to be an expert hacker to launch an attack.

Exploit kits contain a set of exploits and according to user's web browser version they choose the best one to deliver drive-by-download attack. Beyond hacking capa-

bilities which include malware creation and distribution, exploit kits are also capable of managing compromised host by using their command and control servers. Once the machine is infected, it reports back to C&C server and takes commands to execute.

Typically, the interaction between exploit pack and a victim can be summarized as follows:

- Attraction is the phase in which attackers lure the victim into connecting with the exploit kit. This phase is generally carried out by using spam mail, search engine poisoning [49, 50, 51] and infecting legitimate site with malicious links that redirect victims to the exploit pact.

- Fingerprinting is the phase in which exploit pack performs reconnaissance against version of the victim's web browser and plug-ings, operating system and IP address of the victim.

- Exploitation is the phase in which exploit kit chooses the most appropriate exploit code and delivers it to the victim to execute. Once the client application is exploited, it gets additional payloads from exploit pack or its C&C server.

- Persitence is the phase in which exploit kit installs auxiliary modules to hide itself and provides persistence access to the victim.

Currently, there is bunch of exploit kits in the wild. Some of the well known exploit kits and their prices are depicted in Table 2.2.

Table 2.2: The Most Used Exploit Kits and Their Prices

Exploit Kit	Price	Release Year
Mpack [52, 53]	$1,000	2006
Blackhole (v1.0.0) [54, 55]	$700/three months or $1500/year	2010
Gpack [56]	$1,000 - $2,000	2013
Styx exploit pack [57, 58]	$3,000 / month	2012
Cool (+ crypter + payload)[59]	$10,000 / month	2013
Sakura [60]	$1000-2000 / month	2012

According to the 2014 Internet Security Threat Report by Symantec [61], Go1 Pack had a share in 23% of all Web-based attacks in 2013, closely followed by updated version of Black hole exploit kit with 19%. Another exploit kit is Sakura which took 14% of overall kit usage in 2013. The next 2 in the top 5 kits in 2013 are Styx (10%) are Coolkit (8%). (See Figure 2.3)

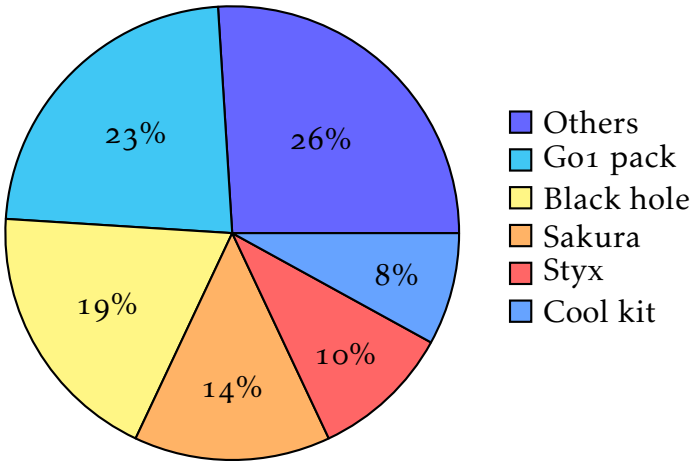


Figure 2.3: Top Exploit Kits in 2014 [61].

2.4.2 Remote Access Tools (RAT)

As stated before, Remote Administration Tool (or Remote Access Trojan) (RAT) is a software aimed at remotely controlling the hacked system. Typically, RAT has ability to capture screen shot, video and keystrokes, sniff network traffic and other malicious actions to theft all safety critical information of the infected system. Though RAT provides technically advanced feature, the user does not have to know background in depth instead they just use graphical user interface to manage infected remote host.

According to Dell Secure Work report on underground hacking market [62], the RAT in 2014 is getting cheaper when compared to the prices in 2013. According the report darkcomet, blackshades, cybergate, predator pain, Dark DDoser are the most popular RAT along the hacker community and they are sold between $20 and $50. Additionally, there are some RATs whose source code were leaked to the community such as CrypterShear, Stewart, Poison Ivy, etc.

2.4.3 Metasploit Framework

The Metasploit Framework [63] is an open source project that allows hackers to create and deploy malware equipped with publicly available security vulnerabilities. Metasploit Framework is generally used in penetration testing to asses an organization's network security capacities. To be more precise, it allows scanning target systems, identifying security vulnerabilities and finally exploit them.

Besides that, Metasploit has the world's largest database of exploits - offers more than 28,000 stable exploits, for local and remote vulnerabilities. Typically, once attacker successfully penetrates into the computer, metasploit framework installs its advanced well-known agent, called meterpreter. This remote shell allows to exfiltrate

confidential information like dumped local users password, migrate into stable process, install additional software, escalate user privileges, pivot onto other systems etc.

Metasploit also supplies automated malware creation functionality as well as build-in obfuscation techniques. To create malicious trojan which enables remote access to the victim computer, one can use **msfpayload** module by choosing the appropriate payload type. Furthermore, the metsaploit framework supports (called **msfencode**) different obfuscation modules to easily evade anti-virus solutions. Metasploit offers the following options for malware developers:

- Generating binary and shellcode - a small piece of machine code generally embedded into the malware to start remote shell.
- Exploiting server and client side vulnerabilities and integrating these exploits into malware sample.
- Bypassing anti-malware solutions, for example exploit framework encodes malware sample to hide from anti-viruses or uses encrypted channels to communicate with the target in order to bypass network-based solutions IPS, IDS and firewall.
- Automatically running post-exploitation attack vectors.
- Creating multiple listeners to command the victim machine.

As open-source version of metaploit is console-based software and contains lots of modules and tools the user needs to know some basic computer, networking and hacking terms and internal structure of metasploit. The usage details and the advanced features of the metasploit framework are mentioned in [64, 65].

2.4.4 Social Engineering Toolkit

The Social-Engineer Toolkit (SET) [66, 67] is an open source project aimed at launching social engineering based attacks. Since it provides menu driven user interface, an end-user can use and launch technically sophisticated attacks without facing any difficulty. However, if the attacker needs to create more reliable and advanced attacks, he has to configure its parameters and install some dependencies. [65] describes the available SET parameters and their aims.

It is important to note that SET attack vector heavily depends on the Metasploit framework which furnishes automatic payload generation and listener setup. SET includes different attack vectors to exploit human as the weakest link of the information system. The main attack vectors provided by SET framework is as follows:

- **E-mail Based Attack:** Creating fake e-mail messages to deceive the victim into performing the requested actions in the e-mail body such as opening malicious attached file which contains an exploit capable of hacking the victim computer.
- **Web Based Attack:** SET provides various web-hacking based social engineering attack vector. One of the most known and used is Java applet attack. In this attack, SET clones a legal web site into local file system and serves this fake website. If the victim browses that page and accepts the execution of the malicious applet his computer will be immediately hacked. Furthermore, SET creates fake web pages which include different browser exploit allowing drive-by-download attack.
- **Malicious Media Generator:** SET framework can also produce malicious storage media such as CD/DVD and USB. Once the victim plugs these devices into his computer, the autorun feature of the Windows OS for storage devices executes the malevolent payload.

2.5 Anti-Malware Analysis Techniques

Malware authors have been employing more and more anti-malware analysis tricks and techniques to overwhelm automatic analysis attempts and to make the malware analysis process too slow and tedious for manual analysis. In this way, malware is becoming more smart and adaptive to survive. These techniques fall into several categories and employ various tricks. In this section, the common anti-malware analysis techniques are described in detail.

2.5.1 Code Obfuscation Techniques

In the context of software development, obfuscation is a way that makes software harder to reverse engineer. To this end, these techniques transform a given program to a different form while preserving its functionality. Initially, these methods were developed to cope with violation of the intellectual property of software products. However, today they have been extremely used by malware creators to evade from detection of anti-virus scanner. In this section, the common obfuscation techniques are briefly described. The details can be found in [68, 69, 5, 6].

Garbage Code Insertion: Malware authors can insert redundant and useless instruction or code to change its binary content without loosing its main functionality. For example, adding NOP (no operation) instructions into different portion of

the code is the first thing coming to mind. On the other hand, a programmer calls useless blocks of code and then roll variables, conditions and registers back to the initial positions.

Instruction Replacement: This method substitutes predefined instructions with equivalent instructions. For example, to clear eax register, malware authors can use one of the following options.

```
mov eax, 0
xor eax, eax
and eax, 0
sub eax, eax
```

Listing 2.1: Alternatives for cleaning eax register

Instruction Permutation: If two successive instructions and their parameters are independent, it is programmatically possible to reorder them without having any impact on program's behavior.

Code Transposition: This technique reorders the blocks of code by using different flow changing mechanism while preserving the behavior of program. It may be done at the level of instructions or modules.

2.5.2 Anti-Virtual Machine Techniques

Today, both system administrators and users prefer VMs because it is easy to rebuild a machine from a snapshot. Following this preference, malware authors realized that virtualization technology is used to dissect malicious executables, and they started to obfuscate their source code with anti-virtual machine tricks. With these techniques, a malware attempts to detect whether it is being run inside a virtual machine or on a real machine. If the virtual machine is detected, the malware can act differently or simply do not run. Issues with reliability and incomplete test information may mislead the analyst. Since Vmware is the most used virtualization platform, this section focuses on anti-VMware evasion techniques. However, these techniques can be applied on other virtualization platform like Virtualbox.

2.5.2.1 Hardware Fingerprinting

Hardware fingerprinting consists in looking for special virtualized hardware pattern unique to virtual machines. For example, the MAC address of the network card, specific hardware controllers, BIOS, graphic card, and so on. These hardwares have

some special names that help to identify virtual machines. Fingerprinting can be carried out using Windows Management Instrumentation (WMI) classes and APIs [70]. The following WMI command, which is executed in powershell, gives the BIOS details of real and virtual machine, (see Figure 2.4).

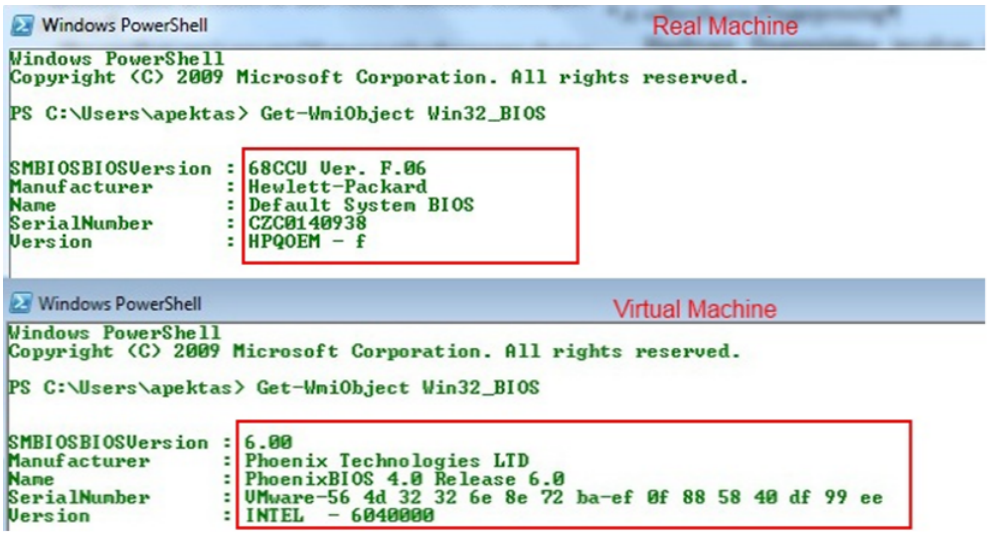


Figure 2.4: BIOS card details: WMI in real and virtual OS, [70].

2.5.2.2 Registry Check

The registry is a centralized, hierarchical database for application and system configuration in Windows operating system. Access to the registry is assured through registry keys, which are analogous to file system directories. In registry system, 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 content through the registry editor, for example using the built-in regedit command in Windows OS. Alternatively, programs can manage the registry through the Windows registry API functions. The registry is stored hierarchically in key/value pairs and contains the following:

- windows version number, build number, and registered users,
- similar information for every properly installed application,
- the computer's processor type, number of processors, memory, and so on,
- security information such as user password policies, log-in type, file/directory access, etc.,

- installed services.

Since the registry includes such a big database, it definitely holds virtual machine specific key/value pair. Based on this information, Tobias Klein's tool ScoopyNG (see for instance [71]) introduced a proof-of-concept code searching for certain keys within the Windows registry to determine whether the machine is virtual or not. For example, some special registry values for VMware machine are highlighted in Figure 2.5.

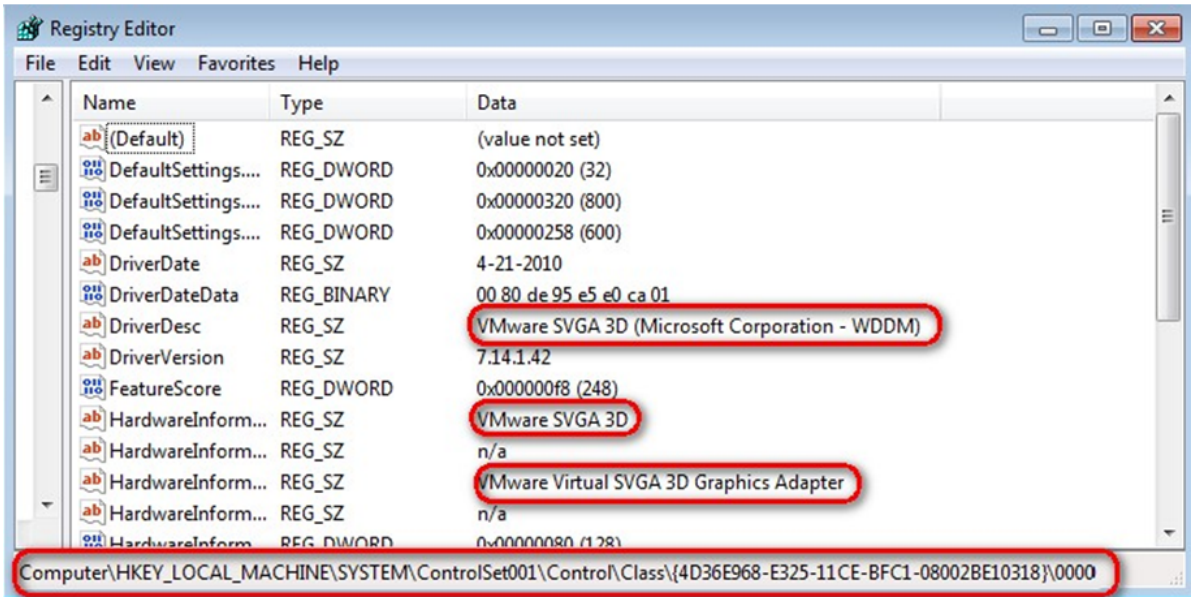


Figure 2.5: VMware specific registry keys on a VMware machine

2.5.2.3 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), or STR (Store Task Register). Actually, most of the malwares with VM detection capability use this technique, which is based on the fact that any machine, whether it is virtual or not, needs its own instance of some registers [72, 73]. Systems such as VMware create dedicated registers for each virtual machine. These registers have different addresses than the one used by the host system, and by checking the value of these addresses, the virtual systems' existence can be detected. A snap code for this technique are given in Listing 2.2. However, this technique succeeds only on a single-processor machine and is not compatible for multicore processors.

```
int swallow_redpill () {
  unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
```

```
  *((unsigned*)&rpill[3]) = (unsigned)m;
  ((void(*)())&rpill)();
  return (m[5]>0xd0) ? 1 : 0;
}
```

Listing 2.2: Snap Code of Red Pill Technique [74]

2.5.2.4 VMware Communication Channel Check

Ken Kato discovered the presence of a host-guest communication channel in VMware so called backdoor Input/Output (I/O) port. VMware uses the I/O port 0x5658 ('VX' in ASCII) to communicate with the host machine. For the interested readers, justifications are available in [75].

There are more commands supported by the backdoor I/O port, such as to obtain data from the Windows clipboard or the speed the microprocessor in the unit of MHz. An example of extracting the VMware version of the virtual machine by using VMware I/O backdoor is given in Listing 2.3. The most common commands are displayed in Figure 2.6. A detailed documentation is available on [76].

```
#define MAGIC 0x564d5868  // VMware backdoor magic value = "VMXh"
#define PORT  0x5658      // VMware backdoor I/O port = "VX"
#define GETVERSION 0x0a   // Get VMware version command id = 10

__try {
  __asm{
    mov eax, MAGIC;
    mov ecx, GETVERSION;
    mov dx, PORT;
    in eax, dx;
    mov test_vmware, ebx
  }
}
__except(EXCEPTION_EXECUTE_HANDLER){;}

if (test_vmware == 'VMXh')
  printf ("VMware Detected!!!\n");
else
  printf ("VMware not Detected...\n");
```

Listing 2.3: Assembly Code to Detect VMware Machine by looking VMware I/O Port

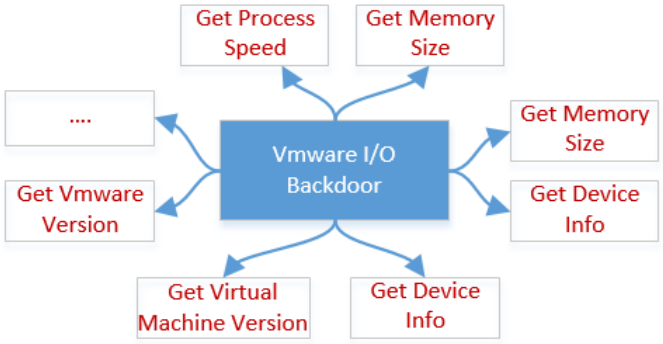


Figure 2.6: VMware I/O Backdoor's Main Functionalities

2.5.2.5 File & Process Check

The VMware environment creates many artifacts on the system. Many VMware specific processes continuously run on the background. Besides that, there are some VMware specific files and folders. Malwares can use these artifacts, which are present in the file system and process listing, to detect VMware. For example, when VMware tools are installed in the guest Windows machine, three VMware processes (e.g., VMwareService.exe, VMwareTray.exe, and VMwareUser.exe) run on the background by default. Malware can detect these processes while searching the process listing for the VMware string. In addition, VMwareService.exe runs the VMware Tools Service as a child of services.exe. VMware can be detected by searching the registry for services installed on a machine or by listing services using the "tasklist" or "net start" command (see for instance the output of the tasklist command executed in the VMware, Figure 2.7).

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 may help to find clues about the existence of the VMware image.

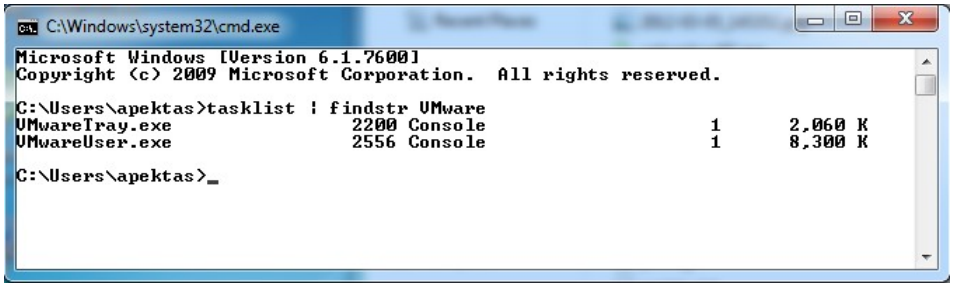


Figure 2.7: Searching VMware Processes in Process List

2.5.3 Anti-debugging

In the context of software development, a debugger is a tool used to detect errors in the code. In the field of malware analysis, debuggers are used to reverse-engineer malware samples and figure out sample's behavior. Since debugging is very beneficial against malware, malware writers employ anti-debugging techniques to hinder debuggers.

Anti-debugging techniques can be divided into four categories:

- **API calls:** Several Windows API functions provides functions to determine if the executable executed by a debugger.
- **Flags:** To stop execution of malware to the specified position, debuggers use flags. Thus attackers take advantage of these flags to figure out the existence of a debugger.
- **Execution time:** Since debuggers run malware step by step, execution time of a sample takes longer. Consequently, malware authors can use execution time of the malware to skip detection.
- **Debugger vulnerabilities:** To prevent debugging, malware authors sometimes modify executable headers which crash debuggers because of their executable parsing vulnerabilities .

Table 2.3 lists and shortly explains the anti-debugging techniques. Details about each technique can be found in [77, 78, 79, 80].

2.5.4 Anti-Disassembly Techniques

In malware analysis terminology, disassembly refers to the process of understanding software source code either by obtaining its pure code or by obtaining assembly equivalent. These tools are called disassembler and widely used by malware researchers while performing manual analysis to grasp malicious methods behind malware sample. Contrarily, malware authors employ anti-disassembly techniques to slow and to complicate the analysis of malware.

Essentially, anti-disassembly techniques add special form of code or data into a program which leads disassemblers to produce incorrect output. Indeed, the main reason of this failure comes from predefined assumptions in disassembly algorithms and malware authors take advantage of these assumptions to deceive these tools. Also, there are more advanced techniques e.g. modifying the data and code section of the executable file to thwart analysis.

Table 2.3: Commonly used anti-debugging techniques and their categories

Category	**Method**
API	IsDebuggerPresent CheckRemoteDebuggerPresent NtQueryInformationProcess ZwQueryInformationProcess OutputDebugString FindWindow
Flag	BeingDebugged flags Ntglobal flags Heap flag
Timing	RDTSC QueryPerformanceCounter & GetTickCount Inserting INT 3 (Interrupt)
Debugger Vulnerabilities	OutputDebugString (OllyDbg)

2.5.5 Packing

Packing or executable packing is another obfuscation method. This method compresses original files which in turn embedded into a new executable along with decompression stub. This type of software is called packer. Packing procedure is illustrated in Figure 2.8. When the compressed file is executed, the decompression stub automatically extracts the executable file and runs it without noticing the computer user. Since packing leads malware to hinder detection of security tools especially signature-based and obstruct reverse engineering attempts, it has become very popular among malware authors. According to AV-Test Corporation, malware authors employ packing techniques in 80% of malware detected by anti-virus vendors [81].

Beyond compression, packers can include anti-debugging, anti-disassembly, anti-virtual machine and encryption components. Compared to other anti-analysis techniques, packers are compact form which provide a significant amount of features instantaneously. Moreover, they are easy to use. As a consequence, malware authors prefer packing techniques to bypass analysis attempts. UPX, AsPack, NullSoft, PE Compact, Themida are the well known packers used by malware writers. Malware researchers have explored different methods and techniques to detect, unpacked and analyze compressed executable [82, 83, 84, 85, 86, 87].

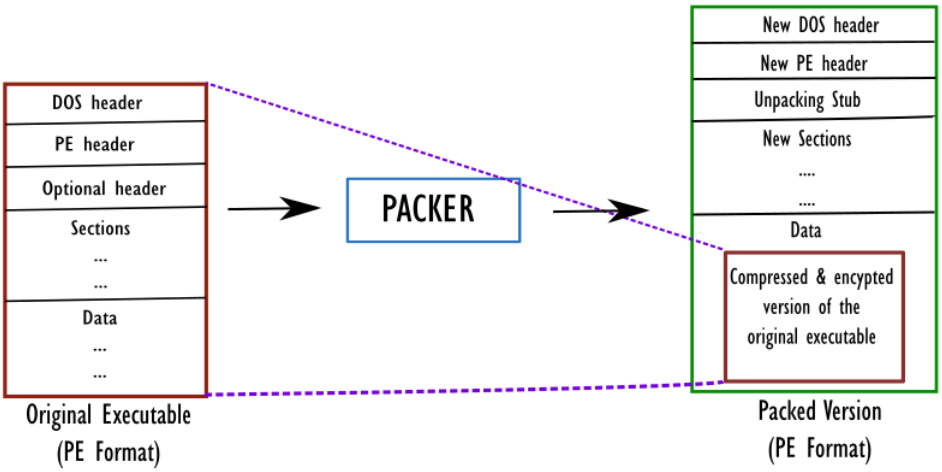


Figure 2.8: Packed executable file format

2.6 State-of-the Art Malware Analysis Methods

Malware analysis is the process of examining and dissecting malicious software to explore and reveal its functionality and objectives. Besides that, it aims at identifying malware spreading mechanisms in order to stop its malicious activity. Currently, the first options in the detection of malicious software is anti-virus tools which use signatures to catch malware and remove them from the system. Indeed, signatures are simply byte sequences or patterns extracted from analyzed file and vulnerable to obfuscation techniques covered in the previous section.

Malware researchers have explored different techniques to detect and mitigate threats coming from malware samples in arm race with malware authors. These techniques can be divided into the three categories which are introduced in following sections (see Figure 2.9).

- **Static Analysis:** Static analysis involves extracting static information (strings, import functions, meta-data, etc.) from binary file to analyze and deduce sample's harmfulness.

- **Dynamic Analysis:** Dynamic analysis refers to the process of executing malware in a controlled environment and monitoring its run-time activities on the host system.

- **Manuel Analysis:** The process of manually reversing the malware source code by using debuggers, disassemblers and other tools.

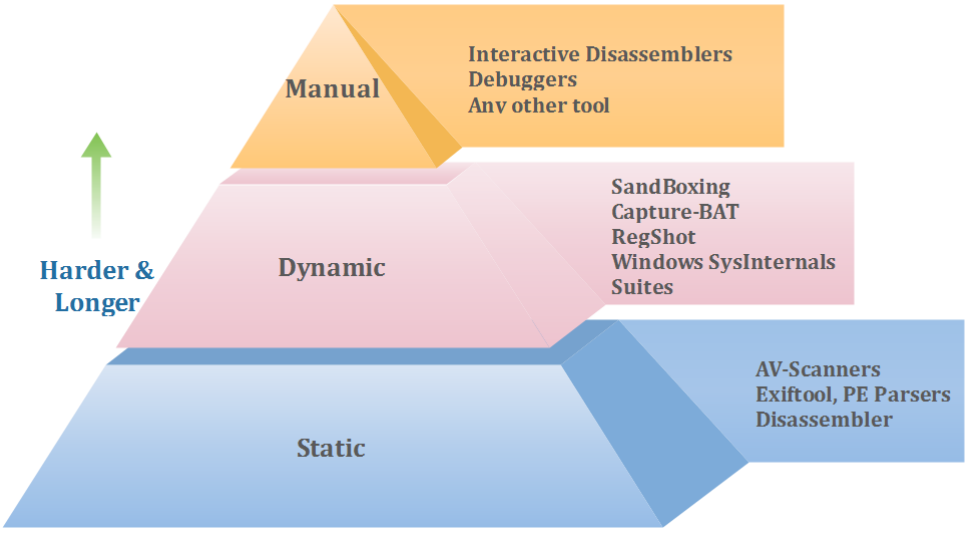


Figure 2.9: Malware analysis pyramid

2.6.1 Static Analysis

Static analysis consists in examining a binary file without running it on the system. Since the source code is not available, malware is disassembled and the created execution paths are analyzed. In static analysis, researchers firstly check possible functionalities of the sample, then create simple network signatures based on the gathered information and finally determine sample's maliciousness. The first step of static analysis consists of looking for obvious indicators of malicious software. This step is similar to the analysis procedure in traditional antivirus where file fingerprint (usually file hashes e.g. MD5, SHA) is calculated and matched with an already-known malware. The second step is the deep file analysis where the file format and content is investigated. The following checks are applied:

File Unpacking: Packing of the file needs to be determined. If the file is packed before deep search, it has to be unpacked and pure executable needs to be obtained. There are different packing methods to obfuscate malware but unpacking procedures are highly sensitive to its former packer and obtaining pure executable code is challenging. However, there are research efforts to create common unpacker for different packers, [88, 89, 86, 90].

Plain Text Matching: The plain text embeded into the executable such as URL, domain name, IP address, output messages are checked and information about being malicious is gathered as much as possible. Generally, strings or grep utility is used to explore plain text in the file.

Anti-Virus Scan: Scanning a given sample with anti-virus engines is the first step of the malware analysis. Sometimes, AV-engines can cause false-positives results meaning that AV-engines labels a sample as malware which is indeed a benign file. Consequently, malware analysts need to be suspicious to that kind of mistakes and cross-check the result by different tools and solutions.

File Meta-data Analysis: By leveraging hidden meta-data of an executable, malware analysists can obtain useful information that help them to draw a conclusion about whether the analyzed file is malware. For example, portable executable (PE) file format contains valuable information such as compilation time, executable name, imported and exported functions, as well as executable sections and sizes.

Disassembly: Disassembled code of the malicious file is used to detect malware samples by using the some statistical approaches like N-gram modeling.

Static malware analysis is fairly straightforward and fast. It ensures security and safety of the computer systems due to its ability of detecting malicious file without execution. But a trade-off exists between its simplicity and accuracy, static detection scheme can be ineffective against sophisticated malware and may miss important malicious behaviors [91].

Aditionally, the obfuscation techniques introduced by malware authors (such as polymorphism, metamorphism, compression, encryption, and run-time packing) render static analysis complicated, time consuming and almost impractical. Therefore, malware researchers explored and developed dynamic analysis methods which are more resistant to the obfuscation techniques.

2.6.2 Dynamic Analysis

Dynamic analysis techniques involve running a malware sample and observe its behavior on the system. As the file under inspection is executed, dynamic analysis evades the code obfuscation and packing techniques. Essentially, malware typically has abilities to create or modify several OS resources on the compromised device to fulfill its objectives. Accordingly, the dynamic analysis consists of monitoring the following behaviors:

Volatile Memory: Malware can overflow buffers and use the abandonned memory locations to gain access to the device. By capturing and analyzing the device memory, it is possible to determine how a malware uses the memory.

Registry/Configuration Changes: Changes in the registry may be an evidence towards dynamic analysis. Malware often changes registry values to gain persistent access to the system.

File Activity: Malware may also add, alter, or delete files. Therefore, by monitoring file activities, valuable information about the malicious behavior can be obtained.

Processes/Services: Malware may disable Anti-Virus (AV) engines to fulfill its functions, jump to other processes to obstruct analysis or install new services to obtain persistent access to system.

Network Connection: Monitoring network connections is the essential part of dynamic analysis to understand the malware's existence. Destination IP address, port number and protocol can be analyzed in order to detect malware's interaction with the command-and-control (C&C) server.

Sequence of API calls: Since API call sequence reflects the behavior of software, it is very handy to model and represent a software with its API calls.

Observing of these behaviors can give valuable information about software's intention, which is difficult to be gathered by other detection schemes. Further, another advantage is that dynamic analysis can be automated with developed framework. In this way it enables large-scale of malware analysis. However, some limitations in detection may occur due to evolving malware characteristics versus anti-dynamic analysis such as anti-virtual machine and anti-debugging techniques.

2.6.2.1 Overview of Existing Dynamic Analysis Techniques

Dynamic malware analysis is an active research topic. Several methods exist for automatically analyzing malicious software behaviors. These can be gathered into three groups [92]:

- **Difference-based:** Analyze differences between two snapshots of the system, one snapshot recorded before the malware execution and the other one after the execution of the file .
- **Notification-based:** Use notification mechanism about the operating system calls triggered by certain events e.g., registry key changes, file/folder modifications.
- **Hook-based:** Hook APIs in user mode or kernel mode to track the changes applied to the system.

Most dynamic malware analysis tools such as Anubis [93], CWSandbox [94], and Norman Sandbox [95] monitor runtime actions performed by malwares. However, Regshot [96], which is used to detect file and registry changes, monitor these changes by snapshotting the sensitive operating system components (e.g. files, folder, registry keys, and so on).

Security mechanism of most sandboxes (e.g. CWSandbox, Anubis, and Cuckoo Sandbox [97]), which are used for running untrusted or malicious programs in a safe environment without risking real systems, present fairly similar approaches to reveal the behavior of the malware. The common task of these tools includes a variety of details on the malware, such as the network activity and the created files during its runtime. These activities can be monitored through its user or kernel space functions.

Analyzing malicious software in user-space helps to obtain high level invoked functions such as enumerating the active processes, finding locked files, and tracing network connections. However, when working in user-space, hidden processes or connections, which are embedded in the kernel-space, can not be detected. To access hidden information in the operating system, analysis tools must have a kernel-space module. These modules need to use kernel space functions to gather hidden information from the user. However, kernel-space analysis requires deep and solid knowledge of Windows OS and it is really a difficult task.

As an alternative to kernel-space analysis, the process of intercepting function calls is a frequently used technique to trace behaviors of the malware and flow of the executable logic. Most dynamic analysis framework such as CWSandbox [94], BitBlaze [98] and TTANalyze [99] (now called *Anubis*) use this technique which is so-called API hooking. The concept of hooking is simple: the call made by an application to a function can be redirected to a custom defined function. API hooking has to be transparent and undetectable by the malware. In case a malware detects API hooking, it may modify its behaviors or jobs in order to be hidden.

From the side of malware, API hooking trap may be avoided by calling undocumented kernel functions directly instead of using API functions. But in case of using kernel functions, the target set to be infected by malware will be limited by the systems whose version of the operating system and service patch level are specified. This situation may create conflicts with the general intention of many malwares about infecting large amount of computer systems.

Moreover, dynamic analysis tools can be grouped under two categories differentiated by implementation platform: analysis in pure software emulator and analysis in virtual machine.

Analysis in Emulator

An emulator is a piece of software that simulates the hardware. A software emulator does not execute code directly on the underlying hardware. Instead, instructions are intercepted by the emulator, translated to a corresponding set of instructions for the target platform and finally executed on the hardware.

For example, Qemu [100] is an open source and full system emulator where the processor and peripherals are emulated by a software. Anubis, Renovo [101] and Hookfinder [102] use the Qemu emulator in order to analyze malware. However, emulator technique is not a straight-forward solution to detect malware. As stated in [103], malware samples can detect the stage of emulation. For example, detecting imperfect emulation of CPU or execution time of the specific commands allows a malware sample to recognize the situation about running in an emulator.

Analysis in a Virtual Machine

Virtualization involves simulating parts of a computer's hardware while most operations still occur on the real hardware for efficiency reasons. Therefore, virtualization is obviously faster compared to emulation. Virtual machines (VMs) also provide dedicated resources like emulators. A snapshot of VM resources (i.e., virtual mass storage, CPU, and memory contents) can be recorded for restoring purposes. This feature can be used to reduce the time required to analyze a malware sample since installation is not required to recreate a clean instance of the analysis environment. For example, CWSandbox uses this technique for analyzing a malware candidate in a virtual Windows environment.

2.6.2.2 Limitations of Dynamic Analysis

Due to the evolving characteristics of malwares, methodologies based on dynamic analysis may pose limitations in monitoring some of them. Malware sandboxes do not take into account any command-line options: they run given executable and monitor its behaviors. Analysis of sample and detection may fail when malware is triggered by command-line. Another drawback of malware sandboxes is their short guard time. In general, sandboxes may fail to wait long enough and they may not record all events. For example, if the malware is set to wait ten minutes before it performs malicious activity, this time-triggered event may not be detected.

Before running, some malware checks the presence of certain registry keys or files on the system, which do not exist in the sandbox. The absence of such legitimate data prevents malware from running inside the sandbox environment. Sometimes,

the sandbox environment may be misled because the malware is created for a specific OS, i.e., the malware might crash on Windows XP whereas it may run on Windows 7. A sandbox can report basic functionality, but it can not classify the file as malicious. The generated reports need to be analyzed in order to conclude whether the file is malicious.

Furthermore, malwares can implement some functionalities to detect VM platforms and they can adapt their behavior. One famous project implementing this detection functionality is the Red Pill project of Joanna Rutkowska. This tool is one of the most known VM aware software, elaborates the feasibility and ease of such detection strategies, [104]. Besides, some tools (e.g. Scoopy [71]) use virtual machine related opcodes or query VM specific virtual registry values.

2.6.3 Manual Analysis

Manual malware analysis refers to the process of interactively examining malware code to gain insights into the behavior of a software and uncover hidden its functionality. Manual analysis requires deep-knowledge and practice in various domains and tools. Malware analyst prefers manual analysis if the static and dynamic analysis fail in detection process or he wants to confirm the obtained findings. Some of the well-known malware analysis tools are listed in Table 2.4.

It is important to note that, generally manual analysis process is tedious and time-consuming. However, manual analysis results are more accurate than other methods. For example, a recent Zeus sample with MD5 bfe6e86081f9c5da45c7eb33272633c4 [128] indicates the importance of the manual analysis. When this sample is executed, it checks the existence of virtualization platform and some dynamic analysis utilities e.g. netmon.exe, procmon.exe. Once it detects such artifacts, this sample changes its behavior and binds shell to TCP port 8000 on the system and waits for incoming requests. If the attacker succeed to connect this port, he can control the computer remotely. Considering today's network security policies especially NAT technology, this behavior totally useless and unreasonable.

As this Zeus sample is packed, the static analysis also impractical to analyze, hence the last choice is manual analysis. Once the process checks are bypassed by debuggers, the sample injects itself into explorer.exe to stay hidden and download an additional executable file and execute it. Then, the downloaded executable adds registry key to be persisted on the compromised system, e.g. after reboot the system malware sample will be again active state. Finally, the sample targets banking information in order to profit. Further information can be gathered from [129].

As mentioned in this chapter each malware analysis method has its advantages

and drawbacks. Malware analysts need to know limits of these methods in order to perform reliable analysis. Figure 2.10 summarizes the pros and cons of the malware analysis methods.

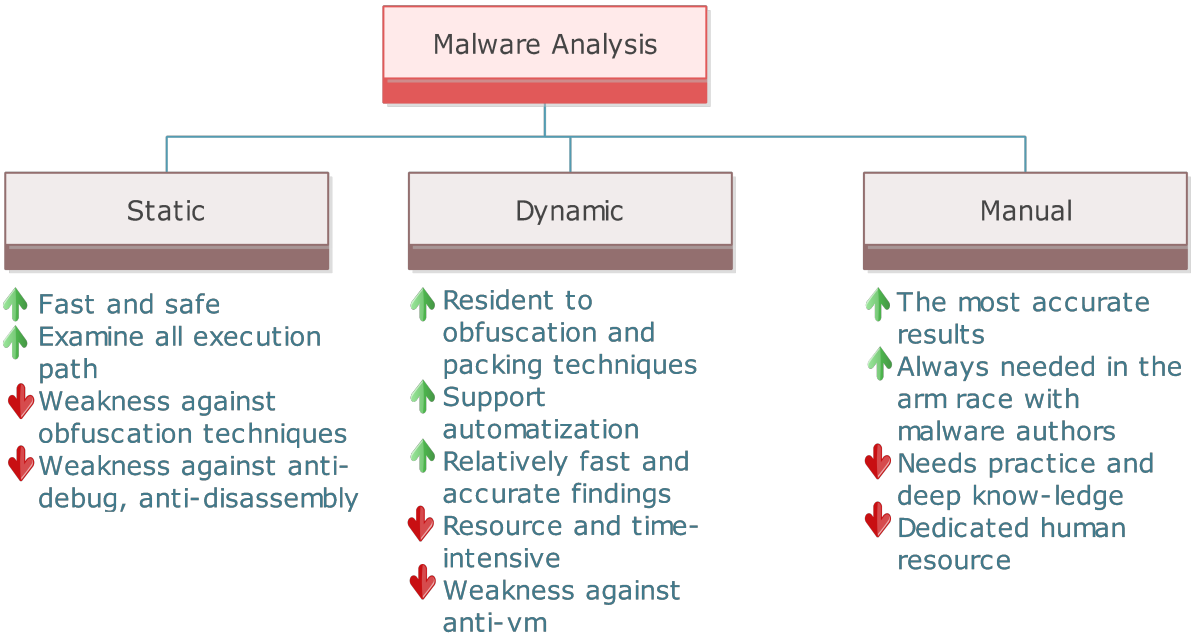


Figure 2.10: Pros and Cons of Malware Analysis Methods

2.7 Summary

In this chapter, we described malware and presented common types of malware. Following that, a quick history of malware evolution was given. We further presented the malware infection methods used to hack computer system in detail. Then we provided various aspects of hacker utilities to create malware and how these utilities and application can be used to launch cyber attack.

We also introduced state-of-the art malware analysis methods and discussed cons and pros of each of these techniques. We further provided list of tools practically and effectively used in malware analysis, thus enabling the reader to be familiar in this research field.

In the next chapter, we present related works in the field of malware research. Particularly, we provide the machine learning methods used in the literature for malware classification and discuss their malware representation methods. Finally, we introduce state of the art dynamic malware analysis frameworks and discuss their limitations.

Category	*Tool*	*Objective*
Static	Virustotal [105]	Free online multi anti-virus scanner
	PEID [106]	Identify most common packers and compilers for PE files
	PEfile [107]	Python module enables problematically working with PE files
	PEStudio [108]	Detect anomalies and suspicious patterns for PE files as well as provide score for indicating degree of harmfulness
	IDApro [109]	Advanced disassembler and debugger that explores software
	Dependency Walker [110]	Shows the imported and exported functions of PE files
	PEview [111], PE-Browse [112], PE Explorer [113]	View and Edit sections of PE files
	Resource Hacker [114]	View and Edit resource section of PE file
	Bintext, strings, grep, findstr	Scan and extract text from any file
Dynamic	Process Monitor [115]	Advanced system monitoring utility for Windows-based system that shows real-time system activities
	Process Explorer [116]	Advanced task manager for Windows system that explores processes and threads
	Autoruns [117]	Explores the auto-start services, drivers, software for Windows system
	Process Hacker [118]	Process Explorer + open source system resource monitoring utility
	Capture-Bat [119, 120]	Driver-based behavioral analysis tool for malicious software
	Regshot [96]	File and registry comparison utility accordingly snapshots
	Volatility [121]	Advanced memory-based forensic analysis tool
	Sandboxes: Cuckoo [97], VirMon [122], Anubis [93], etc.	Dynamic software behavior analysis tools
	Network Sniffers: Wireshark [123], Tcpdump, Netmon [124]	Network packet analyzer tools
Manual	OllyDbg [125], Immunity Debugger [126]	Assembler level debugger for PE files
	WinDbg [127]	Debugger for user mode applications, kernel mode drivers as well as Windows operating system
	IDApro [109]	Advanced disassembler and debugger that explores software

Table 2.4: General Overview of Malware Analysis Tools

Chapter 3

Related Work

This chapter presents previous researches about malware analysis and classification. These research efforts use different malware modeling techniques using static or dynamic features obtained from malware samples. Furthermore, we cover and discuss their modeling techniques and the testing set of these researches.

Over the last fifteen years, significant research has been conducted in the area of information security for detection and classification of known and unknown malware by using various machine learning and data mining techniques. These techniques can be divided into 4 major categories according to their malware modeling approaches:

- **N-gram:** This approach involves a sequence of byte array extracted from various information of a malware samples, such as binary content, disassembled instructions or API calls.
- **Control-Flow:** Control flow involves ordered function calls, instructions and statements that might be executed and evaluated when a program is running.
- **API & System Calls:** In this approach, the API and system calls, which reflect the behavior of a software, are extracted from malware samples in order to model overall behavior of a software.
- **Behavior Abstraction:** It involves expressing malware behavior in terms of high-level concepts for simplicity, conciseness and clarity.

3.1 Malware Modeling Techniques

3.1.1 N-gram

In the field of statistic, an n-gram is a fixed size sliding window of byte array, where n is the size of the window. According to size of n-gram, some special term is used to refer these n-gram, e.g. 1-gram is referred unigram, 2-gram - bigram and 3-gram - trigram. Larger size of n-gram generally referred by the value of n, e.g. 5-gram. For example the “81EDD871" sequence is segmented (represented) into 5-gram as “81EDD", “1EDD8", “EDD87" and “DD871".

N-gram is used in various disciplines and domains like language identification, speech recognition, text categorization, information retrieval, and even information security. Basically, n-gram is used to build a model for given problem, then the model makes prediction based on extracted n-gram features. Accordingly, n-gram modeling have widely used in machine learning problems.

In the early literature, researchers use fix size n-gram and variable length n-gram extracted from binary content of the analysis file and opcodes obtained after disassembling for the purpose of presenting benign and malicious software. To detect and classify samples, they applied various machine learning algorithms including SVM, decision tree, naïve bayes, etc. and also ensemble learners (also known as boosting) on these n-grams.

The first known use of machine learning in malware detection is presented by the work of Tesauro et al. in [130]. This detection algorithm was successfully implemented in IBM’s anti-virus 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 [131]. 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 shown that the developed method performance was not sufficient. Schultz et al. has used machine learning methods in [132]. Function calls, strings and byte sequence were used as a feature set. 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. [133] 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. Tests have been done with malware and 40 benign files. With this set, test results shown 98% of success. Using the data in [133], the accuracy slightly dropped to the 94% level.

Kolter et al. [134] used 4-grams as features and selected top 500 n-grams through information gain measure. They applied naive Bayes, decision trees, support vector machines(SVM) and boosting algorithms. Boosted decision tree outperformed all other methods and gave promising results such as receiver operating characteristic(ROC) curve of 0.996.

In our previous study [135], 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. In our study, 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. 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.

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

the classification accuracy for training and testing is achieved their highest success percentage of 99% and 98%, respectively.

Shafiq et al. come up with a data-mining based malware detection method that uses distinguishing static feature extracted from portable executable (PE) files [136]. They follow a threefold methodology: (1) extract structural and useful features from PE files, (2) preprocess data (i.e. remove irrelevant, redundant and noisy information) and (3) apply classification algorithm.

Their feature set consists of 189 features obtained from field in the PE file structure such as section headers, DLLs referred, resources, etc. To improve quality of feature vector and reduce the processing overheads, they used three well-known feature selection filters; Redundant Feature Removal (RFR), Principal Component Analysis (PCA) and Haar Wavelet Transform (HWT).

They evaluated their method on two malware set, VX Heavens [137] and Malfease [138] datasets which contain about 11.000 and 5.000 malware respectively by using five machine learning algorithms; IBK, J48, RIPPER, SMO and NB. J48 outperforms other algorithms in terms of the detection accuracy and achieves 99% detection rate with a less than 0.5% false positive rate. They claimed that their method is robust to different packing technique but they did not conduct any experiment on runtime packer that can modify all PE structure.

Wessenegger et. al. [139] extract n-gram from the payload of the each network packet to detect intrusion attempt in network level. Similarly, PAYL [140], McPAD [141] and Anagram [142] analyze byte n-grams for detecting server-side attacks. Basically, these studies also employ n-gram features but differ when considering the weight of each vector. For example, McPAD and Anagram use term frequency while Anagram employs boolean frequency. Besides that, the methods Cujo [143] and PJScan [144] use token n-grams with a binary map for identifying malicious JavaScript code in web pages and PDF documents, respectively.

3.1.2 Control-Flow

Control Flow Graph(CFG) or Program Dependence Graph(PDG) semantically represents the structure of program. Essentially, CFG is constructed by scanning basic blocks of a given program. Then this generated graph is used to identify the similarities of software, based upon the similarities of the common sub-graph of the software.

Bonfante et al. proposed a malware detection strategy based on control flow graph composed of six kinds of node jmp, jcc, call, ret, inst, and end (fundamental x86 assemble instructions) which correspond to the structure of control flow [145]. The constructed graph presents the all execution paths that might be traversed through a pro-

gram while running. To reduce graph, the authors removed inst and jmp nodes from the graph and linked all predecessors to its unique successor.

In their study, they employed two algorithm. The first algorithm searches for exact match between the CFGs of know program and the program under test. The second algorithm checks isomorphism between the reduced CFG of the program under test and training programs. Experiments on 2278 samples collected from public sources show similar false positive rates, 4.5% for first algorithm and 4.4% for second algorithm. The authors did not used other metric to measure the effectiveness of their study.

Cesare et. al. propose static malware detection method for malware variants by using similarity search over set of CFGs [146]. In this study, two feature sets are used to characterize a program: subgraphs of size k and any character sequence of size q (q-gram) constructed from CFG. While constructing feature vector of a program, the set of most of the 500 most frequent features are selected from training samples. Consequently, the information of rare features are lost due to this feature reduction.

To calculate similarity of two programs based on their feature vector, Manhattan distance is preferred because of its (runtime) computational efficiency when compared to the more traditional ones such as Euclidean. They evaluated introduced malware classification system with 15398 real malware samples collected from honeypots and 1601 benign binaries obtained from Windows system directory and the Cygwin executable directory are used.

The authors select the values of k and q as 4 and 10 respectively since it is recommended in previous studies. Additionally, the threshold value of 0.6 was empirically chosen by authors. The evaluation shows 1% false positives rate. Though authors used unpacking technique to remove obfuscations before building feature vector, effectiveness of their emulation-based unpacking method is questionable in malware domain.

Ding et al. presented a control flow-based method to model executable behaviors [147]. Before building opcode behavior, they utilized IDA Pro tool to decompile executable. Then they constructed flow graph of a program that represents behavior of a program and traverse the graph to grasp all possible execution paths. Following that, they used n-gram method over all execution path to obtain features.

To reduce feature space, they employed information gain and document frequency. The authors preferred to normalize feature vector in order to have unit length. In their study, they used the well-known supervised classifiers, namely, the K-Nearest Neighbor (KNN), decision tree and Support Vector Machine (SVM) to classify a program as malware or not.

They conducted experiment consists of 650 benign executables and 650 malicious executables, all of which are in Windows PE format. The benign executables are se-

lected from the building system directory of Windows XP. The malicious executables were collected from netlux.org and Offensive Computing [148]. They used 5-fold cross validation to evaluate the performance of each classifier. Their experimental result shows the average accuracies of DT, KNN, SVM are 93.2%, 92.0%, and 88.0%, respectively. The main shortcoming of their study is that they focused on unpacked executables.

Similar to the control flow-based approaches presented above, Park et al. proposed a malware classification schema based on maximum behavioral subgraph detection [149]. A behavior graph is created from the system calls captured during the execution of the suspicious software (in a sandboxed environment). The similarity between two behavioral graphs is calculated by maximal common subgraph measure. If two behavioral graphs have a small distance than a pre-determined threshold, these samples are considered as similar behavior and attributed to the same malware family. The method has been evaluated on a set of 300 malware instances in 6 families. In the paper, however, the authors only provided the class-wise accuracy for the tested families.

3.1.3 Application Programming Interface

Application Programming Interface(API) calls is an interface allowing computer user to access and request OS actions/services, such as creating a process, writing a file, establishing network connection, etc. For this reason, sequence of API calls requested by an executable program could be used to characterize its activity or behavior.

There are two methods to obtain API calls list: static analysis (e.g. IDA Pro disassembly tool) or dynamic analysis (e.g. API hooking). The main problem in extraction of the function or API calls through static analysis by disassembler (e.g. IDA Pro) is that software can include multi execution paths, dirty and unused codes. Moreover, disassemblers can be evaded by anti-disassembly methods. These drawbacks make call graph improper for further use. Instead, if there is no run-time protection, one can accurately obtain API call list through dynamic analysis.

Salehi et al. introduce a malware detection method relying on the API calls and arguments list [150, 151]. To this end, malware samples are executed in a virtual machine until all processes terminated or execution time reach timeout of two minutes and their behaviors monitored by WINAPIOverride32 tool which provides Windows API calls invoked during analysis along with their argument and return values.

To reduce the number of features, the frequency of the API calls smaller than specified threshold T is omitted. In their study, they used T as 20 for malware and 10 for benign files. Then, well-known feature selection algorithm ReliefF is applied. Follow-

ing that, the binary feature vector of a sample is created by the presence of selected API list in a given sample.

To prevent over-fitting problem the authors employed 10-fold cross validation procedure. They used well-known classifier such as Random Forest, J48, FT, SMO, NB, HyperPipe. They achieved the best accuracy 97.4% when applied Meta-classifier AdaBoostM1 in combination with J48. As authors used pre-defined set of API calls and upper-bounded feature space with a constant value T to represent a software, which may produce loose of important behavioral information. Moreover, they utilized limited amount of samples in evaluation phase.

Chandramohan et al. proposed malware detection framework that relies on the modeling interaction between software and security-critical OS resources [152]. These resources include file system, registry, process, thread, section, network and synchronization. To collect runtime behavior of malware samples they used sandboxing technology but they did not mentioned the name of the tool used in their study. After obtaining behavior report, feature extraction is applied. Their feature extraction consists of three steps:

- Identify OS resources presenting in the behavior analyze report.
- Group related actions corresponding to the OS resources obtained in Step 1.
- Repeat Step 2 until all OS resources identified in Step 1 is finished.

In contrast to existing approaches whose feature space grows proportionality to the number of malware sample, they used upper bound $N = 16,652$ to summarize behavior of sample based on the pre-defined actions on OS resources. Similar to [151] binary feature vector is created in which n^{th} element is assigned as '0' or '1' based on the presence or absence of that feature. Thus, each program either malware or benign is presented by an N-dimensional binary feature vector.

They conducted an experiment involving 5,300 malware and 100 benign samples by using Support Vector Machine and logistic regression algorithms. In the experiment 5-fold cross-validation, the initial malware set are randomly divided into 5 subset and training and testing is performed 5 times, is used. LR classification algorithm achieves 99.6% detection accuracy and 1% false positive rate.

3.1.4 Abstraction

Bayer et al. propose a clustering approach to identify and group malware based on behavioral similarity [153]. To this end, they first perform dynamic analysis to acquire the execution traces of malware programs. Then they generalize execution traces into

behavioral profiles that model the activity of a software in more abstract form. More precisely, they create the behavioral profile of a software by abstracting the following artifacts.

- System calls
- Control flow dependencies
- System call dependencies
- Network activities

In dynamic analysis phase, they introduced taint analysis to Anubis [93], one of the automated malware analysis system, in order to collect the listed features in the above. After that, authors model and define a program's behavior by following form of action list. Here, += operator is the increment operator and malware behavioral profile composes of collection of the OS events.

```
Malware behavior profile += (OS object type, object name, OS operation,
    operation's attributes, operation's status)

OS object types  = {file, registry, process, job, network, thread,
              section, driver, sync, service, random, time, info}
```

Following this behavior definition, they transform feature set in a suitable form for the clustering algorithm. For clustering process, in their study, they utilized Locality-Sensitive Hashing (LSM) based algorithm. They clustered the set of 75,692 samples in 2 hours and 18 minutes which is quite long and achieved precision(the probability for an estimated instance as class c to be actually in class c) and recall(the probability for a sample in class c to be classified correctly) of 0.979 and 0.980 respectively.

Bailey et al. also abstract a malware's behavior and create a behavioral fingerprint in order to categorize them [154]. The proposed fingerprint is composed of system state changes such as files written, processes created, etc. rather than in sequences or patterns of system calls.

To measure similarity of the groups of malware, they construct a tree structure based on single-linkage clustering algorithm. Basically, this algorithm defines the distance between two clusters as the minimum distance between any two members of the clusters. They tested their method over real world malware samples(including samples that have not seen in the wild, thus these samples did not have a signature to match) and obtained better classification results than anti-viruses.

Similarly, Jang et al. present BitShred, a large-scale malware clustering technique for automatically exposing relationships within clusters [155]. The authors used feature hashing to reduce high dimensional feature space and to mine correlated features

between malware families and samples. The proposed method can utilize any analysis reports that produce boolean set of features, or extract boolean features from report files. They computed malware similarity by using Jaccard and BitVector Jaccard distance. Besides that, they also developed a parallelized version of BitShred within the Hadoop framework.

To create a reference clustering data set, they used different anti-virus labels provided by VirusTotal [105]. For larger-scale experiment, they used 131,072 malware samples and obtained precision of 0.942 and a recall of 0.922. This task took about 27 minutes.

Rieck et al. introduced a classification method that aims to determine whether a given malware sample is variant of known malware family or is a new malware strain [156]. They capture system call traces and represent the monitored behavior of malicious software by means of special representation called Malware Instruction Set (MIST) which is inspired from instruction sets used in CPU. In this representation, the behavior of a sample is characterized with a sequence of instructions.

More precisely, MIST instructions are composed of three fields: a CATEGORY field, an OPERATION field, and optional several ARGBLOCK fields. CATEGORY field contains 20 different category and each of those groups a set of related operations. Additionally, the ARGBLOCKs for each category widely varies between categories. The authors optimized this representation for data mining algorithms.

They employed behavior reports obtained from the CWSandbox [94] dynamic analysis platform for their testing set containing 10,072 samples. They applied the Support Vector Machine (SVM) algorithm to identify malware classes. The experimental results shows 88% accuracy.

Mohaisen et al. proposed a malware classification system, named CHATTER, based on ordered high-level system events [157]. CHATTER consists of four major steps as follows:

- Sandboxed execution: Though their method does not only rely on one sandbox, they prefered AUTOMAL [158] to collect runtime artifacts of sample under analysis. While the tool provides information for various activities they made use of only network features.

- Behavioral documents: The output of their tool named AUTOMAL mapped into an alphabet. In this way, the behavior of sample abstract into a format which is more operable by machine learning algorithms.

- Utilizing n-grams: They enumerate and count all unique n-grams in a behavioral document obtained in previous step. These n-grams are utilized to represent

malware samples.

- Machine learning component: They applied supervised machine learning algorithm over constructed n-gram feature vectors.

To evaluate their study, they employed three well-known machine learning algorithms; the k-nearest neighbor (k-NN), support vector machine (SVM), and decision tree classifiers. The testing set consists of the malware families: 1025 instance of Zeus, 544 instance of Darkness, and 1130 instance of Shady RAT(SRAT). They manually identify and label training data by the help of the expert analysts. They achieved 80% accuracy and when they paired with base-line classifier consists primarily of file system feature the accuracy reached 95%. Though they obtained encouraging result on testing set, their method is not suitable for malware sample that does not perform network activities.

3.2 Dynamic Malware Analysis Tools

As stated in Chapter 2, dynamic analysis requires executing a given program that is being analyzed and monitoring its run-time activities in a control environment. Dynamic analysis consists in observing the activities of the suspicious file while allowing their execution in a controlled environment. These systems track and inform about file, registry, network, and process activities. To successfully detect malware and take appropriate counter measures, dynamic analysis can be considered as an integrated scenario and solution of an environment provided to malware for being deployed and performing its tasks.

Over the last ten years, malware researchers have been proposed and developed various tools and methods addressing the problem of malware analysis. Before discussing the current dynamic malware analysis frameworks in the following subsections, it would be good to indicate survey paper which gives an overview of dynamic malware analysis and lists their strengths and drawbacks [159].

3.2.1 Anubis

Anubis [93] formerly TTAnalyze [99], is an emulation-based dynamic malware analysis system. It performs analysis operations on Windows XP OS running on Qemu emulator. During analysis, it monitors Win32 and native API functions and their parameters through Qemu. With process monitoring feature, Anubis can only monitor all processes created by file under analysis and omit other running processes.

Anubis monitoring relies on comparing dynamically the instruction pointer of the emulated CPU with previously known entry points of functions in shared libraries. Since the system runs on an emulator, duration of analysis may be longer compared to the analysis duration in real PC and this anomaly can be detected by malware.

3.2.2 CWSandbox

CWSandbox [94] leverages API hooking technique in user mode to track malware's activities. CWSandbox executes given sample either in a real PC or in a virtual Window machine. Once the sample is loaded into memory, API hooking is performed by inline code overwriting. The sample is executed in a suspend mode and then all loaded DLL's API functions are overwritten. During this initialization step, CWSandbox identifies the exported API functions and injects necessary hooks. Hence, CWSandbox collects all called functions and their related parameters.

To hide the indicators of the running environment i.e. CWSandbox from a given sample, CWSandbox hooks API calls which provides these indicators. After finishing analysis, CWSandbox creates a high-level report about activities from which malware analyst can quickly follow them. Since it collects data in user mode, low level operations and undocumented function calls can not be captured.

3.2.3 Cuckoo

Cuckoo [97, 160] is an open source analysis system and relies on virtualization technology to run a given file and supports major virtualization platforms such as ESXi, Virtualbox and KVM. Cuckoo is written by Python language and allows users to create their modules.

Cuckoo can analyze both executable and non-executable files. For example, if an office document is fed into the system, then Cuckoo uses an end-user application such as Microsoft Word or OpenOffice to open it and reports run-time activities of this document. These activities including pre-defined Win API functions and their parameters are monitored and captured by its user-space API hooking technique. Furthermore, Cuckoo has active community whose members share signatures and modules to the researchers.

Owing to the fact that analysis module of Cuckoo runs at user level, malware can easily notice presence of the analysis attempt causing to change its behavior. Besides that, malware analyst might need time to understand Cuckoo's structure and working mechanism.

3.2.4 Capture-BAT

Capture-BAT [119, 120] is another dynamic analysis tool developed by New Zealand chapter of honeynet.org [161]. Capture-BAT monitors process, registry, and file activities at kernel level, and it captures network traffic using winpcap library. Furthermore, it offers selection of events through its filtering interface that can be used by the analyst to prevent noisy events to be captured. Since Capture-BAT is not an automated malware analysis system, serious concerns exist on whether it can efficiently handle the high penetration of new and existing malware.

3.2.5 Norman Sandbox

Norman Sandbox [95] is a dynamic malware analysis solution which executes a given sample in virtual environment that simulates a Windows operating system. This simulated environment emulates host computer and network and to certain extent Internet connectivity. Norman Sandbox transparently replaces all functionality that is required by a sample under analysis with custom-made versions.

Norman redirects all networking requests originated by sample under analysis to the simulated servers in order to capture them. For example, if a sample tries to resolve a host name to IP address, related DNS query forwarded to the simulated DNS server instead of public DNS servers. This procedure is not noticeable by the sample. Other network services such as HTTP, SMTP, etc. are redirected to the equivalent servers.

Instrumenting the APIs enables effective function call hooking and parameter monitoring. The observed behaviors (e.g., function calls, and arguments) are stored to a log file. Furthermore, Norman Sandbox also monitors auto-start extensibility points (ASEPs) that can be used by malware instances to ensure persistence and automatic invocation after shutdown - reboot sequences.

3.2.6 Dynamic Malware Analyzer

In our previous study, we developed and deployed Dynamic Malware Analyzer (DMA) tool to analyze anti-virtual machine aware malware samples in VMware environment [162]. DMA focuses on anti-virtual machine evasion techniques to provide secure and reproducible environments for malware analysis and its implementation issues. Malware is identified based on their behaviors by taking precautions related to the anti-virtual machine detection techniques.

DMA employs Pin tool [163, 164], free tool provided by Intel for dynamic instrumentation of programs to hidden from detection attempts of VM-aware malware. DMA tracks Windows API calls to change outputs of these calls. For instance, if OS's

response contains the string "VMware", the control passes to the proposed replacement routine where the returned value is changed to a more appropriate value such as "Microsoft" or to a value that would have been returned on a host Windows OS. In a similar way, when VM specific instructions such as SIDT are in the course of being executed by the sample, the control passes to replacement routine where the value of the destination operand is set to a value that would be obtained on the host Windows OS.

.Net framework is used as the underlying platform and DMA enjoys a user-friendly graphical user interface. Before using DMA, DMA needs to be configured through its simple settings' interface. DMA can monitor anomalies occurring on the system through checking out all processes, connection table, service details and file activities on Windows operating system. Success ratio of detection is tested by using public malware sets with an accuracy of 92%. Though DMA is great tool for estimating the harmfulness effect of the malware file to the system, it lacks of user-independent and automatic analysis. Table 3.1 compares features of the major dynamic malware analysis frameworks.

3.3 Discussion and Conclusion

In this section we review malware detection and classification approaches based on data mining and machine learning. Flowing that we briefly present state of the art dynamic malware analysis frameworks. There are several important observations can be made from the literature review done in this chapter, that are of relevance for this research:

- The representation of malware by using n-gram profiles has been presented in the open literature. In these studies some promising results towards malware detection are presented. However malware domain has been evolving due to survivability requirements. Thus, malware tries to evade anti-virus scanners to perform its functions. Obfuscation techniques have been developed in order to avoid detection by anti-virus scanner. And these techniques disturb n-gram features of binary form of the malware used by the previous work.

- Call-graph comparison according to graph edit distance(GED) is NP-hard problem. In other words, calculating GED is more complex and computationally expensive.

- In the literature, there are studies that employ only static features like byte sequence, op-code sequence, printable stings and API call sequence. Though these

Table 3.1: Comparison of the major dynamic malware analysis frameworks

	Anubis	CWSandbox	Norman Sandbox	DMA	Cuckoo	Capture-BAT	VirMon
Analysis implementation							
User-mode component		✓		✓	✓		✓
Kernel-code component		✓				✓	✓
Virtual machine monitor							
Full system emulation	✓			✓	✓	✓	✓
Full system simulation			✓				
Analysis targets							
Single process	✓	✓	✓		✓	✓	✓
Spawned processes	✓	✓			✓	✓	✓
All processes on a system				✓	✓	✓	✓
Complete operating system						✓	✓
Analysis support for							
API calls	✓	✓	✓		✓		
System calls	✓	✓	✓		✓		
Function parameters	✓	✓	✓		✓		
File system operations	✓	✓	✓	✓	✓	✓	✓
Process/thread creation	✓	✓	✓	✓	✓	✓	✓
Registry operations	✓	✓	✓	✓	✓	✓	✓
Instruction trace				✓			
Networking support							
Simulated network services			✓				
Internet access(filtered)	✓	✓	✓	✓	✓	✓	✓

methods effective in detection and classification of unpacked malware samples, they would be ineffective on obfuscated malware samples.

- In the proposed studies, there is an inevitable trade-off between "curse of dimensionality" and "poor interpretability". In other words, on one hand if feature-space increases the analysis framework becomes infeasible. On the other hand, once the feature space is reduced, important information might lose which causes to reduce the accuracy of the analysis.

- To the best of our knowledge, dynamic analysis systems still use old versions of Windows OS, for instance Cuckoo merely employs Windows XP and 7, as an underlying analysis environment. However, computer owners generally prefer to

upgrade their OS to the newest version. Therefore, existing systems may fail at analyzing malware targeting new versions of Windows OS. Reliability and availability of malware analysis scheme may be a major concern subject to the release of new Windows OS version. Consequently, the next generation of dynamic malware analysis solutions should be adaptable to the future versions of OS.

Chapter 4

VirMon: A Virtualization-Based Automated Dynamic Malware Analysis System

This chapter presents our dynamic malware analysis platform to explore behavior of a malware and describes how our platform differs from other approaches.

In this chapter, we present Virus Monitor (VirMon) as a scalable automated anti-malware system designed to be robust versus malware targeting new versions of Windows OS. The features of VirMon are as follows:

- **Portability:** VirMon can use any version of Windows OS including XP, 7, 8, 8.1 and even 10 beta as the OS of the analysis machine. When a new OS is released, it can be easily adapted.
- **Scalability:** The analysis capacity, i.e., the average number of analysis per minute, can be increased by connecting new virtualization servers to the existing system. VirMon capacity can be improved by adding new analysis machines upon the increase of the analysis workload.
- **Network virtualization:** The network traffic of analysis machines is distributed to different network locations via VPN to masquerade their IP addresses. This decentralized design approach ensures that the analysis system is not detectable by malware's network level precautions such as comparing public IP addresses of the analysis system.

The rest of the section is organized as follows: Section 4.1 presents the network visualization method. The implementation details of VirMon components and their functionalities are elaborated in Section 4.2. In Section 4.3 two real-world malware samples are used to illustrate the effectiveness and analysis results about monitoring the malware activities. Finally, some conclusions are given in Section 4.4.

4.1 Network Virtualization Infrastructure

Malware authors try to evade dynamic analysis' detection scheme by using the publicity and popularity of dynamic malware analysis tools. As a primary task, they collect private information about analysis systems. For instance, public IP addresses of malware analysis systems constitute the most important private information. If the public IP addresses of the environment in which malware is being executed, are a known address by malware authors, malware changes its behavior or simply refuses to run on these systems. For example, AV Tracker (see [165] for instance) is a web platform publishing public IP addresses of well-known analysis systems. In order to cope with malware's IP-based protection capability, a network virtualization infrastructure relying on VPN technology is designed. Its main aim is to mask the IP addresses of the analysis machines. The core nodes of the analysis system and their functionalities are designed as follows:

- **Network virtualization:** It forwards network traffic to virtual analysis environment.
- **VPN Server:** It matches sensor & analysis machine.

4.1.1 Sensor Device

Sensors are embedded fan-less mini-PC placed (shown in Figure 4.1) at any location on the Internet in a plug-and-play fashion. The main purpose of these distributed sensors is to send analysis machine's traffic to the Internet and receive back the responses through a secure channel. They are used to mimic the analysis machine as if it is directly connected to the Internet. IP layer information (for instance, source & destination IP and TTL) of all network packets originating from analysis machine is altered on sensor clients. Following this network logic, when a malware tries to grasp public IP address of its outgoing channel, it obtains sensor's public IP address.

On these sensors, a lightweight version of FreeBSD, a powerful open source OS, is running along with VPN-client application. Each sensor has configuration files and a public IP address assigned by the related organization to connect back to the data

Figure 4.1: Sensor device

Table 4.1: Sensor specifications

CPU	500 MHz AMD Geode LX800
DRAM	256 MB DDR DRAM
Storage	CompactFlash socket, 44 pin IDE header
Connectivity	3 Ethernet channels (Via VT6105M 10/100)
I/O	DB9 serial port, dual USB port
Board size	6 x 6" (152.4 x 152.4 mm)

center. Indeed, these are elementary but safety-critical settings that need to be secured and well protected. To prevent removal of these settings in case of power failure or OS crash, they are statically written on the file system and mounted read-only mode. Once the device is plugged into an electric power, OS automatically boots and connects to the VPN Endpoint.

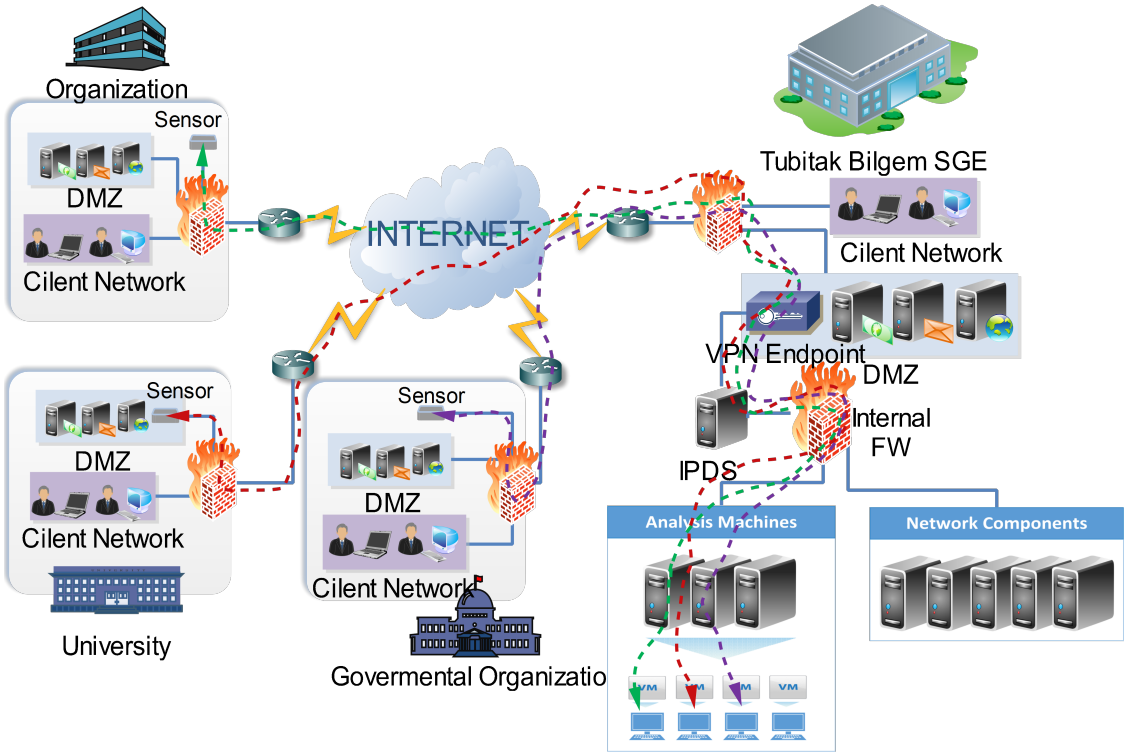


Figure 4.2: The topology of VirMon: local and remote components of the presented dynamic malware analysis system

4.1.2 VPN Server

After a successful VPN connection, an individual sensor is matched with an analysis machine. Following this configuration, all network packets are transferred via this particular sensor. Figure 4.2 illustrates different locations of the sensors in different LANs, sensor devices can be deployed in the DMZ or in the separated network segment of the organization. For VPN connection, sensor devices need a public IP address and an open outgoing 443/TCP port on the firewall enabling access to the VPN-server. Sensor devices do not create any additional security concerns due to the access restrictions in the organization LAN, for instance, sensor clients can be only accessed through SSH from the data center.

Individual static C class network is assigned to each analysis machine on the VPN Endpoint. When a suspicious file while being analyzed in the system tries to make a connection to a remote host, VPN Endpoint identifies the related sensor and forwards its network traffic to that particular sensor. Hence, the private (local) IP address of an analysis machine is mapped to the sensor's public IP address. This transparency is illustrated in Figure 4.3.

In addition, to improve transparency of the system, separate VLANs are deployed for each analysis machine on the internal firewall and virtualization server. Hence, analysis machines are isolated from each other's traffic and broadcast domain is reduced.

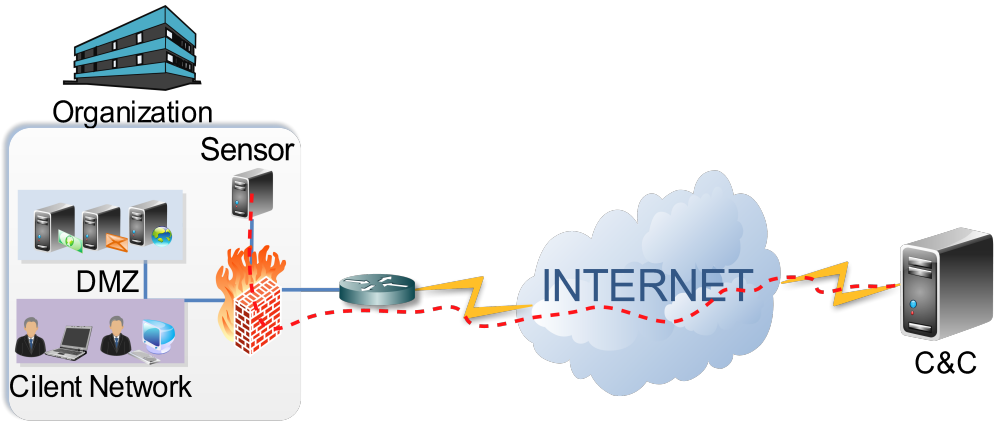


Figure 4.3: The logical topology of the system - The C&C server perceives analysis machine as if it is working behind the firewall.

4.2 Design of VirMon

In this section, we elaborate VirMon dynamic analysis system components and their functionalities. First, we present the collection methods of behavior-based activities of the analyzed suspicious file with special emphasis on the merits of kernel callback mechanism. Then, we describe the network topology of VirMon where network activities of the analyzed sample are automatically gathered.

I-) Analysis machine components include mini-filter driver and driver manager. They are responsible for reporting host-based process, registry, and file system activities performed by the analyzed file, see Figure 4.5.

II-) Network components are responsible for reporting network activities of the analyzed file. The functions are realized by the following client and servers (see Figure 4.4):

- the sensors at different locations that forward network traffic of analysis machines to the command and control (C&C) servers,
- a virtualization environment running the analysis machines,
- an application server managing whole analysis processes,
- an IDS generating alarms, extracting files and HTTP requests from network traffic,
- a DNS server replying and logging all DNS requests,
- a NTP server used to synchronize all machines in the system,
- a relational database storing all events captured during analysis.

4.2.1 The Components of Analysis Machine

Each machine dedicated to analyze malicious software, stated as the analysis machine, has two main components: an application working in user space is responsible for managing analysis process and a kernel driver. The application running in user space is used to create a communication link between the analysis machine and the application server. The file taken from the application server is first executed by the user space component, and the process id information obtained by executing the file is sent to the kernel space component. Then, the kernel space component watches the three main activities of this process: process, registry, and file system.

Activities of the process (or processes) initiated by that executable and (if any) child processes (i.e., additional process created by some of these processes) are monitored

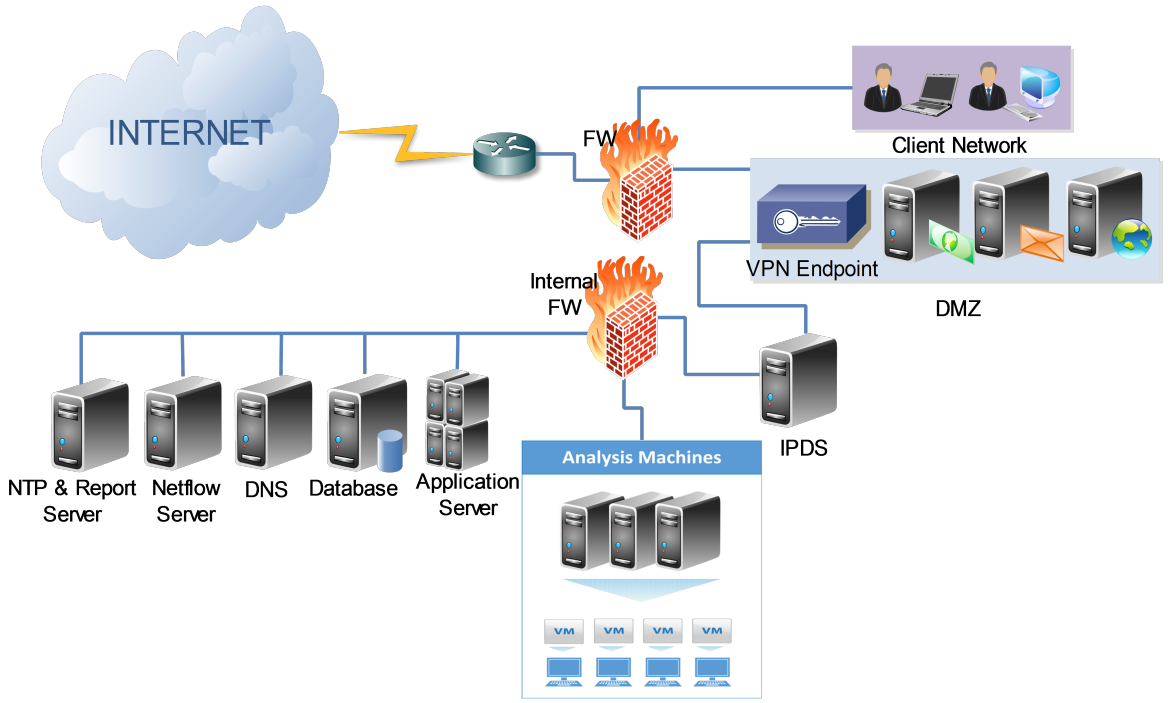


Figure 4.4: The topology of VirMon: local and remote components of the presented dynamic malware analysis system

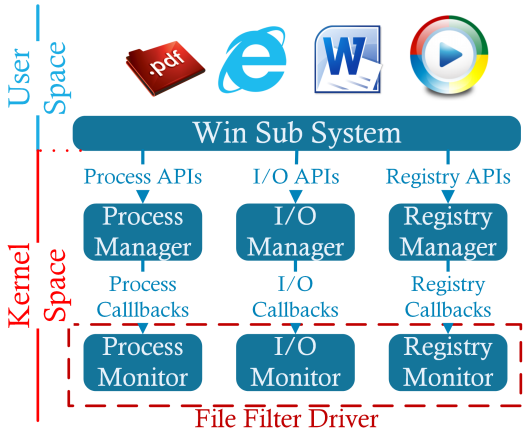


Figure 4.5: Overview of the analysis machine components (e.g., filter drivers)

by means of the kernel callback functions being embedded in kernel driver. These components of analysis machine are plotted in a top-down approach in Figure 4.5. The advantage of the callback mechanism and the technical details about how to monitor run-time activities are presented in the following sub-section.

4.2.1.1 Windows Callback versus API Hooking

API hooking is one of the preferred methods for dynamic malware analysis. In API hooking, the function calls are instrumented and redirected to a predefined function

where they are logged. During hooking, all function calls and their parameters are logged. This allows malware researchers to analyze the behavior of the file.

There are two types of API hooking: user and kernel space. In user space API hooking, the Win32 and native Windows API functions are considered. One of the deficiencies of this method is that the applied processes are not transparent to malware, and malware can change its behavior and mislead the results.

On the other hand, kernel space API hooking relies on injection of code into kernel. It is important to note that even though kernel patching was a popular method in the past, probability of causing fatal errors, i.e., system crash, was high. To mitigate this issue, Microsoft introduced a kernel protection mechanism (patch guard) with Vista 64 bit OS to prevent it from unauthorized operations [166]. When a change attempt occurs, this mechanism causes a blue screen of death (BSOD) to preserve kernel integrity and it makes kernel hooking impractical.

To overcome these limitations, we use kernel callback mechanism, which provides detailed view of run-time events based on defined conditions on a system basis [167]. To be able to use callbacks, a kernel driver needs to be built. However, this is a very challenging task since deep knowledge and skills about kernel programming is required to be able to build the driver. Basically, this driver, also known as mini-filter driver, intercepts all IRP requests made by an application and decides whether allowing or refusing these operations according to the given rule set. The portability is followed by the claim of Microsoft, which states that kernel-callback mechanism is reliable and compatible with all versions of Windows including their 64 bit versions.

4.2.1.2 Process Monitoring

Malware creates new process or changes an existing one in order to run its actions without being detected. It may create services, which are normally run under privileged account to manipulate safety critical information on the compromised host. Thus, process activities of a given sample are very useful and crucial at analyzing run-time behavior.

To obtain the information about run-time events provided by the callback mechanism, related functions need to be called with their relevant parameters. For process monitoring, the "PsSetCreateProcessNotifyRoutine" function allows the mini-filter driver to monitor changes applied to the running processes. When an application initiates any process or thread activity, OS provides its parent and child processes to the developed driver. However, in some situations the thread creation may be misleading and detailed analysis is required. For example, when there is a working process being among those processes tracked by the driver, and if the created thread does not belong

to this working process, this event is also recorded as a remote thread creation. On the other hand, in the event of deleting a process, notification routine runs in the context of the last thread exiting from the process. Consequently, finding the unique identifier (i.e., PID) of the killer process becomes difficult. Under these circumstances, deletion events of the processes tracked by the driver are simply recorded and unique matching with a particular deleting process is not guaranteed.

4.2.1.3 Registry Monitoring

Windows registry is a hierarchical database used to store configuration information of applications, hardware and users, (see [168]). Many malware uses Windows registry to gain persistent access to the system, for example, they register themselves to the registry to be executed automatically at startup. In addition, registry can be used to prevent users from calling the task manager, Windows defender utility, and so on. Last but not least, it allows the user to run services for remote connections such as remote desktop, it disables firewall to allow any access including malicious software. Attackers can use these registry keys to gain authority over the OS and persistent access to it. In this context, there is a strong need to monitor registry to understand malicious behavior.

When an event occurs on the registry, "CmRegisterCallback" function can provide related information to the mini-filter driver. To track registry events, one needs to identify which actions to be monitored in the driver via some self-explanatory constant values (e.g., *RegNtPostCreateKey*, *RegNtPreDeleteKey*, *RegNtEnumerateKey*). In VirMon, registry operations about *OpenKey*, *CreateKey*, *DeleteKey*, *SetValueKey*, *DeleteValueKey*, *QueryValueKey* and *EnumerateKey* are monitored.

4.2.1.4 File System Monitoring

Malware copies itself or its variants to various locations in the file system and then adds a registry key to start automatically while booting. While some of them download extra payloads and update themselves periodically to hinder detection, others may enumerate sensitive information from compromised hosts to exfiltrate. In addition, it can alter legal programs to reach at their malicious objectives. For example, it can create shortcut icon on the desktop with malicious arguments.

"FltRegisterFilter" function along with its callback actions can be used to monitor file system activities on the system. Like registry monitoring, the actions to be tracked have to be addressed in the driver accompanied by some constant values (e.g., *IRP_MJ_WRITE*, *IRP_MJ_READ*, *IRP_MJ_QUERY_INFORMATION*). In VirMon,

to avoid redundant and distracting file operations, we consider only read, write, and delete events performed by the tracked processes.

4.2.2 Network Components

Malware needs to connect very often to the C&C servers to send confidential information collected from compromised machines or to receive C&C servers' commands. This bi-directional communication makes analysis of the network activities of malware as an inevitable requirement to be fulfilled by malware researchers. In VirMon, we use different network solutions, such as VLAN, VPN, IPDS, and firewall to monitor network activities of suspicious files. We are motivated by the previous analysis results about malicious activities in the national network (see [169] for instance), in which malware samples have been collected via high interaction honeypots [170, 171].

4.2.2.1 Virtualization Infrastructure

The open source Oracle VirtualBox [172] is chosen as the virtualization infrastructure to host and to deploy malware analysis machines. VirtualBox supports both 32 and 64 bit CPU. VirtualBox also provides accessibility features such as remote machine management, display of multiple remote machines via web interface, and offers command-line interface (VBoxManage) for automated tasks.

Via the command-line interface, selected tasks of malware analysis machines including automatic reloading from the safe state, setting network configurations, management of user accounts and services are remotely commanded. The virtualization infrastructure is designed to have multiple physical servers, simultaneously. The scalability of analysis machines ensures high performance of the virtualization infrastructure. Explicitly, the number of analysis machines deployed in the physical machine is kept in proportion to its number of cores. For example, suppose s units of the physical servers exist, each server has p number of processor and each processor has c cores, then the total number of the analysis machine, denoted by N, is calculated by:

$$N = s \times p \times c \tag{4.1}$$

4.2.2.2 DNS Server

The Domain Name System (DNS) is a dynamic database service for translation of Internet domains and host names to IP addresses. To bypass network restrictions, network protocols can be extended and used for transmission of malware's data. For example, through DNS tunneling a file can be transferred. Although data transfer

rate is low, it can still present a threat to an organization while being less remarkable among other data transfer protocols [173, 174, 175].

To monitor DNS requests made by analysis machines, we manually configure the DNS server address of each machine and forward their DNS queries to our DNS server located in the datacenter. Meanwhile, in order to prevent malware from resolving domain name by using public DNS server (e.g., to prevent the usage of 8.8.8.8 as a well-known Google public DNS server, we redirect all DNS queries to the internal DNS server by using our internal firewall). This way, all DNS requests made by malware can be answered and logged by VirMon's DNS server. Then, this log file is parsed and the extracted information is sent to the database.

4.2.2.3 IPDS Frameworks

Intrusion Prevention and Detection System (IPDS) is a network security solution monitoring network traffic and system activities [176]. In VirMon, an IPDS is introduced to prevent possible networks attacks caused by suspicious files in the system. This secure scheme fulfils the analysis requirement of the malware analyzer by giving an opportunity to acquire all network events including the requested web pages, downloaded files by malware. This statement is supported by the two evolving behaviors of the malware:

1. Nowadays, IT departments allow their users to make only HTTP/HTTPS connections because of security reasons. Consequently, malware has adapted to HTTP/HTTPS protocols to communicate with its C&C server.

2. As a well known fact, malware may download additional contents over Internet such as payloads providing advanced features like dumping user credentials, pivoting to secure networks.

Suricata

Suricata [177], an open source IPDS solution, can prevent malicious attacks such as distributed denial of service (DDoS), port scanning and shell codes. It can also extract files and HTTP requests from live network traffic. In order to circumvent network attacks caused by malware sample under dynamic analysis, we use Suricata as one of the IPDS component.

Bro

Bro is an open-source (comes with BSD license) powerful network analysis framework. Bro is different from the typical IPDS since it can not block the attacks and

does not rely on network signatures but it enables monitoring all network traffic. It supports well-known network protocols, extracts related information from network packets and exposes network activities at high-level [178, 179].

In the hierarchy of VirMon, Bro runs on IPDS server and reads network interface. To extract files from live network traffic, a custom script compatible with HTTP, FTP, IRC, SMTP protocol is created. The developed script logs hostname, URL, filename, file type, and transport layer information (e.g., IPs and ports) to a file which is parsed periodically for storing these information in database. Meanwhile, if the files extracted by Bro Engine have not already been dissected beforehand, they are queued into application server's priority queue.

4.2.2.4 Netflow Server

Netflow is the network protocol, and it generates IP traffic statistics for the given network interface. Netflow analysis report involves the information about source and destination hosts, used protocols, ports, flow duration and size of the transferred data.

Recently, researchers have proposed various malware detection methods based on Netflow, [180, 181]. Motivated by these results, network flow for analyzing malware is considered to be a completing protocol. To collect netflow information, all network traffic made by the analysis machines is duplicated by the internal firewall and forwarded to the netflow server. Then, it extracts brief information about the flow and stores them in the database on a periodical basis.

4.2.2.5 Application Server

The application server is responsible for the management of the malware analysis processes. It assigns an analysis machine to the submitted file. It collects the file activities from analysis system components. Then, it formats the collected data and stores them in a database. After the analysis operations are completed, the application server commands the analysis machines to be restored to a clean state.

4.3 Deployment

The VirMon system is deployed by following the network topology illustrated in Figure 4.4. We have ran the evaluation of VirMon for 18 months. Environment is set up by using HP-blade systems containing 16 physical blade servers and each server has Intel Xeon 6-core 2.93 GHz processor, 16 GB memory and 512 GB internal storage. Since the IPDS and database server requires more computing power with respect to other network servers, they are installed in a separate blade server (see Table 4.2). Other

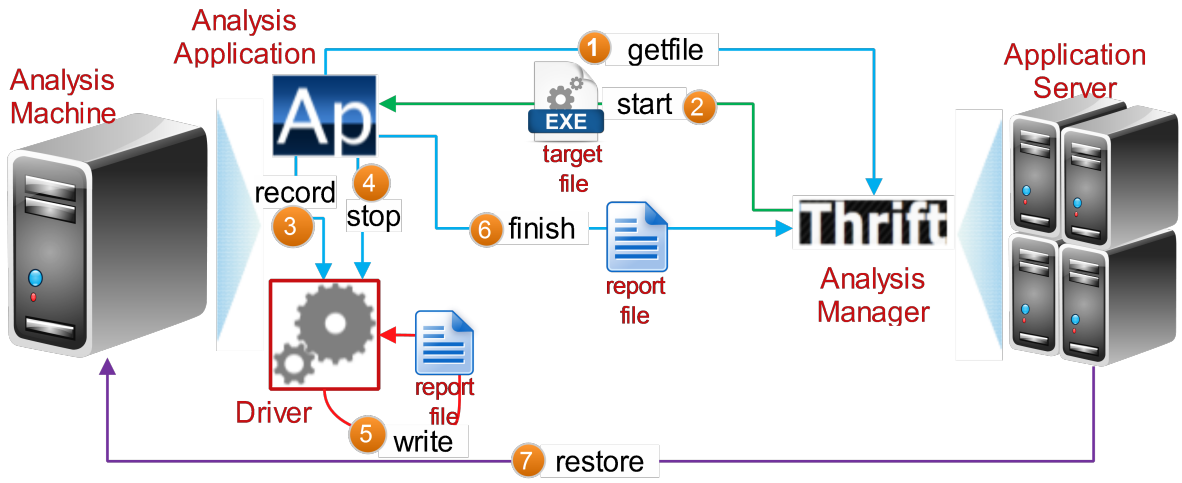


Figure 4.6: High-level information flow and interactions between application server and analysis machine

servers are deployed in the virtualization infrastructure. Total number of the analysis machines is obtained by subtracting the remaining number of required servers: $N = 14 \times 6 \times 2 - 4 = 164$.

Table 4.2: Blade Server Configuration

Blade ID	Blade Count	System
1	1	IPDS server
2	1	Database server
3-16	14	Virtualization servers

4.3.1 The Procedure of Analyzing A Sample File

The procedure followed by VirMon to analyze a submitted file is described. First, the communication mechanism between the application server and analysis machines is illustrated. Then, analysis results of a recent malware sample (i.e., a variant of the hersperbot) targeting online banking users in Turkey are elaborated.

Figure 4.6 illustrates the interaction between the application server and the analysis machine. The process steps are as follows:

1. Analysis application sends a request to the application server to start a new analysis process.

2. The application server chooses a file in its priority queue and sends it to a analysis machine.

Table 4.3: Important run-time activities of a trojan

Time	Event	Process	Detail
26/08/2014 2:48:44.423	Create Process	C:\Windows\explorer.exe	fatura_874217.exe (analyzed sample) MD5:186C097B9D85B3501EFCC4D8D374AFE1
26/08/2014 2:48:55.790	Create Process	%Desktop%\fatura_874217.exe (analyzed sample)	fatura_874217.exe (analyzed sample)
26/08/2014 2:48:55.920	Terminate Process	fatura_874217.exe (analyzed sample)	-
26/08/2014 2:48:56.264	Create Folder	fatura_874217.exe	%APPDATA%\yseszpkf
26/08/2014 2:48:56.269	Create Folder	fatura_874217.exe	%APPDATA%\Sun
26/08/2014 2:48:56.468	Create File	fatura_874217.exe	%APPDATA%\yseszpkf\yqoletyz.dat
26/08/2014 2:48:56.687	Create File	fatura_874217.exe	%APPDATA%\Sun\yqoletyz.bkp
26/08/2014 2:48:57.299	Create Process	%Desktop%\fatura_874217.exe (analyzed sample)	C:\WINDOWS\system32\attrib.exe
26/08/2014 2:48:57.301	Terminate Process	%Desktop%\fatura_874217.exe (analyzed sample)	-
26/08/2014 2:48:57.542	Create Process	C:\WINDOWS\system32\attrib.exe	C:\Windows\explorer.exe
26/08/2014 2:48:57.882	Terminate Process	C:\WINDOWS\system32\attrib.exe	-
26/08/2014 2:48:58.063	DNS Query	C:\Windows\explorer.exe	followtweetertag.com
26/08/2014 2:48:59.272	Send Data	C:\Windows\explorer.exe	https://followtweetertag.com(possibly download config files)
26/08/2014 2:49:01.120	Create File	C:\Windows\explorer.exe	"yqomswoc.bkp, ajukiveq.bkp, yqoletyz.bkp under %APPDATA%\Sun"
26/08/2014 2:49:01.715	Create File	C:\Windows\explorer.exe	"ajukiveq.dat, cfovpdiq.dat, oquthmjk.dat, yqoletyz.dat, yqomswoc.dat under %APPDATA%\yseszpkf"
26/08/2014 2:49:02.012	Create File	C:\Windows\explorer.exe	C:\windows\esem\ohotuzuf.exe (MD5: D082B6AD2F24040E6D651D271823D51C)
26/08/2014 2:49:02.114	Create Reg Key	C:\Windows\explorer.exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\qzofpbuk=C:\windows\esem\ohotuzuf.exe
26/08/2014 2:49:02.345	Send Data	C:\Windows\explorer.exe	https://webislemx.com (for further commands)

3. The analysis application injects codes into explorer.exe process. This process executes submitted file in the suspended mode and then explorer.exe writes PID of this recently created process to a shared memory. Subsequently, the analysis application reads PID info from the shared memory, sends a message involving this information to the driver and a request about recording the events initiated by this process.

4. The analysis application waits until the analyzed process exits or a timeout of 3 minutes occurs. Then, it sends a message to the driver to stop recording events.

5. The driver stops recording and writes collected events to a log file.

6. The analysis application sends the log file to the application server for parsing,

7. The application server parses the log file and stores it in the database. Finally, it reverts the analysis machine to the clean state.

Malware samples used for evaluation purposes were obtained during recent malware campaigns targeted to Turkey. We analyze a trojan, named as hesperbot, which was detected on August, 2014 [182]. This trojan is focused on stealing banking account information to be used towards unauthorized money transfers. The attackers social engineer victims to execute attached files by sending e-mail which looks like it is originated from one of the service providers in Turkey. The analysis of hesperbot is done automatically on a Windows 8 Ultimate 64-bit OS.

Since the number of events including system dll file and registry accesses gathered from VirMon for this malware is too high (10000+), only important events occurred on the system are displayed in Table 4.3. The intention of the malware sample can be easily derived from this table. When the sample is executed, the process created by explorer.exe creates a new process entitled "fatura_874217.exe" having the same name in its directory. In Turkish, "*fatura*" means "*the bill*". This technique, named as process hollowing, (see [183, 92] for instance), has been recently used by malware to hide itself. Then, the process created by explorer.exe terminates itself. The

fatura_874217.exe process created by hollowing technique, creates %APPDATA%\Sun and %APPDATA%\yseszpkf directories and drop randomly named binary files under them. To be hidden, the fatura_874217.exe process creates new explorer.exe, which in turn is used to carry out remaining activities, download configuration files from C&C server, drop new executable and writes it to the auto-start line in the registry, respectively. This analysis shows that the VirMon dynamic malware analysis system successfully collects the run-time behaviors of the file sample.

4.3.2 VirMon Compatibility on Windows 10 beta

In order to show that VirMon (e.g., its mini-filter driver) is adaptable to the newest version of Windows OS, we conducted functional testing of developed mini-filter driver on Windows 10 beta. This test is based on the analysis results of the recent malware sample, known as cyrptolocker, a variant of ransomware, that encrypts sensitive documents on the infected machine and forces to pay a ransom to make them usable again. Since other system components of VirMon work independently, it has been sufficient to install the developed mini-filter driver into the analysis machine to integrate new OS.

Accordingly, we successfully installed VirMon's mini-filter driver on Windows 10 beta without any need to modify or build driver's code. Table 4.4 shows the summarised run-time activities of the cyrptolocker still observed by VirMon during its analysis. Cyrptolocker malware [184] is active at the time of writing this thesis, December 2014. This malware's activities show that it uses the process hollowing technique as in the previously analyzed sample (i.e., hesperbot sample). Then, it probably searches specific file types under C:\drive to encrypt and makes them unusable unless one does not have the description key. Finally, it sends some information to its C&C server and asks ransom from its victim user.

4.4 Conclusion

In this chapter, the virtualization-based dynamic malware analysis system and its components are presented. A mini-filter driver is built to monitor run-time activities of the file to be analyzed. Since kernel-callback scheme is reliable and compatible with all versions of Windows OS, analysis machine can support all Windows OS versions. In addition to its portability feature, the design supports virtualization and scalability, for instance the average rate of analysis can be adjusted by the virtualization approach, and malware armed with IP-based evasion technique can be avoided by the decentralized sensor architecture.

Table 4.4: Run-time activities of the cyrptolocker on Windows 10 beta

Time	Event	Process	Detail
29/11/2014 16:15:53.478	Create Process	C:\Windows\explorer.exe	%DESKTOP%\fatura_892738105.exe (MD5: 76387075C90533AAD14E82A5D94E8486)
29/11/2014 16:15:53.678	Create Process	%DESKTOP%\fatura_892738105.exe	%DESKTOP%\fatura_892738105.exe
29/11/2014 16:15:53.755	Terminate Process	%DESKTOP%\fatura_892738105.exe	-
29/11/2014 16:15:53.964	Create Folder	%DESKTOP%\fatura_892738105.exe	%APPDATA%\ytivyteqyfypequs
29/11/2014 16:15:53.973	Create File	%DESKTOP%\fatura_892738105.exe	%APPDATA%\ytivyteqyfypequs\01000000
29/11/2014 16:15:54.055	Create Process	%DESKTOP%\fatura_892738105.exe	%WINDOWS%\explorer.exe
29/11/2014 16:15:54.888	Terminate Process	%DESKTOP%\fatura_892738105.exe	-
29/11/2014 16:15:54.906	Create File	%WINDOWS%\explorer.exe	%WINDOWS%\whdhufel.exe (MD5: 76387075C90533AAD14E82A5D94E8486)
29/11/2014 16:15:54.925	Create Reg Key	%WINDOWS%\explorer.exe	HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\ebotigob=%WINDOWS%\whdhufel.exe
29/11/2014 16:15:55.162	Create File	%WINDOWS%\explorer.exe	%APPDATA%\Microsoft\Address Book
29/11/2014 16:15:55.162	Create Process	%WINDOWS%\explorer.exe	%SYSTEM%\vssadmin.exe
29/11/2014 16:15:55.198	Create File	%WINDOWS%\explorer.exe	%APPDATA%\Microsoft\Address Book\user.wab
29/11/2014 16:15:55.508	Terminate Process	%SYSTEM%\vssadmin.exe	-
29/11/2014 16:15:59.066	Create File	%WINDOWS%\explorer.exe	"02000000, 03000000, 04000000, 05000000 under %APPDATA%\ytivyteqyfypequs\"
29/11/2014 16:15:59.466	DNS Query	%WINDOWS%\explorer.exe	IT-NEWSBLOG.RU
29/11/2014 4:15:59.786	Query Directory	%WINDOWS%\explorer.exe	C:*
29/11/2014 4:15:59.900	Create File	%WINDOWS%\explorer.exe	Start to encrypt all files located under C:*
29/11/2014 16:16:00.068	Send Data	%WINDOWS%\explorer.exe	https://IT-NEWSBLOG.RU
29/11/2014 16:16:01.951	Send Data	%WINDOWS%\explorer.exe	https://IT-NEWSBLOG.RU
29/11/2014 16:16:02.859	Send Data	%WINDOWS%\explorer.exe	https://IT-NEWSBLOG.RU
29/11/2014 16:17:23.564	Create Process	%WINDOWS%\explorer.exe	C:\Program Files\Internet Explorer\iexplore.exe
29/11/2014 16:17:24.863	Create Process	C:\Program Files\Internet Explorer\iexplore.exe	C:\Program Files\Internet Explorer\iexplore.exe
29/11/2014 16:17:26.933	Send Data	%WINDOWS%\explorer.exe	https://IT-NEWSBLOG.RU
29/11/2014 16:17:27.186	Send Data	%WINDOWS%\explorer.exe	https://IT-NEWSBLOG.RU
29/11/2014 16:17:27.186	Send Data	%WINDOWS%\explorer.exe	https://IT-NEWSBLOG.RU

Two recent malware samples captured in the wild are analyzed to illustrate the portability feature and analysis success of the presented dynamic analysis system. The activities of the analyzed samples are extracted accurately and details of each activity are given with the timestamp, event and process description, which enhances readability of the analysis.

However, since the presented design is based on virtualization technology, it is not capable of analyzing malware sensitive to virtualized environments. Moreover, it can only analyze executable files. In the future, we plan to improve the presented design by adding the analysis skills oriented towards malware sensitive virtualized environment by collecting network events on low kernel level.

Chapter 5

Classification of malware using its behavioral features

After having introduced VirMon dynamic malware analysis system, in this chapter we explain the malware classification process, from feature selection to the classification algorithms used in evaluation phase.

The proposed malware classification method consists of three major stages. The first stage consists in extracting the behavior of the sample file under scrutiny and observing its interactions with the OS resources. At this stage, the sample file is run in a sandboxed environment. Our framework supports two sandbox environments deployed: VirMon [122] and Cuckoo [97]. During the second stage, we apply feature extraction to the analysis report. The label of each sample is determined by using Virustotal [105], an online multiple anti-virus scanner framework consisting of 46 engines. Then at the final stage, the malware dataset is partitioned into training and testing sets. The training set is used to obtain a classification model and the testing set is used for evaluation purposes. An overview of our system including its main functionalities is presented in Figure 5.1.

The remain of this chapter is organized as follows. This chapters begins with describing the dynamic analysis frameworks employed in our research and answers the question of why we choose these tools. In Subsection 5.2.1 we provide an introduction and overview of the current state of the malware behavior sharing formats. Based on these sharing formats, we propose and outline our behavioral features to represent malware. Following that, we describe the Jubatus framework, its architecture, key functional components, and its data conversion method. In Section 5.3, we define

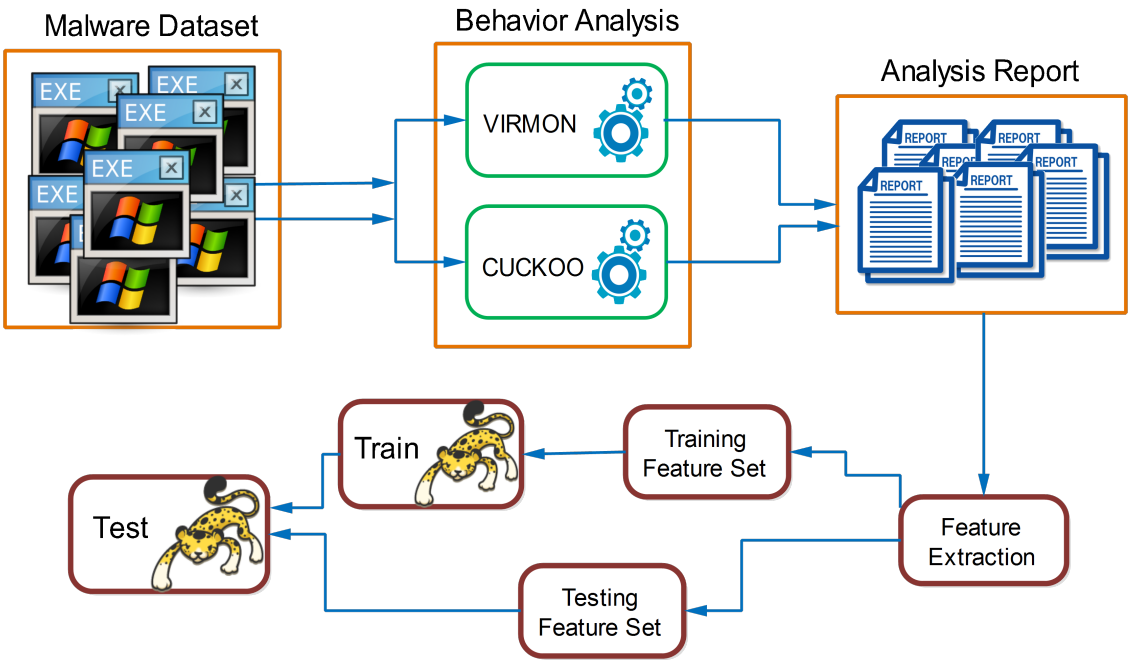


Figure 5.1: Overview of the proposed malware classification system

the online machine learning and provide some details of the online classification algorithms used in our experiments. In particular, we present their mathematical details. Finally, we conclude the chapter by summarizing its main contributions.

5.1 Automated Dynamic Analysis

In this section, the details of the dynamic analysis frameworks (i.e. Cuckoo and VirMon) used in this work are provided, especially the system designed to analyze more samples in a given time with Cuckoo is presented. In this work, we used both of the API calls and system changes to model the runtime behavior of a given file. To this end, VirMon and modified version of the Cuckoo Sandbox is used to automatically obtain behavioral activities. Table 5.1 shows the adopted outputs of the VirMon and Cuckoo for presentation of a software behavior. Following to the brief overview of the dynamic analysis frameworks, we will focus on the feature set.

5.1.1 VirMon

As stated in previous chapter (i.e. Chapter 4), VirMon monitors any system changes occurred on the analysis machine through its Windows kernel level notification routines. During the analysis, the state changes of the OS resources such as file, registry, process/thread, network activities and IDS alerts are logged into a report file. One

Table 5.1: Adopted features from dynamic analysis frameworks

	Mutexes	Process Tree	IDS Signatures	DNS Requests	HTTP Requests	File Activities	Registry Activities	Service Activities	IRC Commands	API calls
Cuckoo	✔	✔	✘	✔	✔	✔	✔	✔	✔	✔
VirMon	✘	✔	✔	✔	✔	✔	✔	✘	✔	✘

of the most important feature of VirMon is automatic analysis that makes malware analysis handy.

Obviously, in dynamic analysis some critical API call set expose the aim of a software. These calls are one of the useful and efficient features to model malware. As VirMon does not provide API calls made by the dissected file, open source Cuckoo sandbox is used to overcome this limitation.

5.1.2 Cuckoo

Cuckoo, an open source malware analysis framework, reports artifacts with its agents while executing a file in an isolated environment. Besides that, it obtains function calls of the file under analysis through hooking method.

We modified Cuckoo in order to increase the number of concurrent analysis it can perform. The number of the guest machine in one analysis server is adjusted depending of the host system used for analysis. For example, 10 guest machines run in a server containing 32 GB memory and 4×4 Xeon CPU. Windows XP SP3 is selected as analysis OS since it consumes less memory and CPU power. Simultaneously, we employed five host machine which has the same configuration with each other and each of them severing 10 guest machine to dissect malware.

It is certain that malware might sometimes infect itself into other internal or external networks, attacks hosts located on the Internet or takes commands from Command and Control servers(C&C). Therefore analysis environment needs to be organized properly in order to cope with that kind of attacks. For this reason, virtual network interfaces (i.e. VLAN interfaces) are defined on the each host server and on these interfaces proper rules is applied to separate network traffic between each machines.

Furthermore, to allow each analysis machine to access to the Internet, NAT configuration is performed on cuckoo host server. On the other hand to restrict access to the

internal subnets iptables rules are applied. In addition, malware analysts can simultaneously monitor ongoing analysis and can interact with the process in any time.

The curl utility is used to automatically submit files into Cuckoo. Though the analysis report can be saved into either relational or non-relation database, we prefer to save into file system. Due to the extensive storage need, one host machine is connected to the 2 TB storage units. This storage formatted as glusterfs file format and shared between other host machines. Refer to the Section A.2 for the step-by-step Cuckoo installation instructions for the latest Ubuntu server (e.g. 14.04 LTS).

5.2 Feature Extraction

5.2.1 Malware Behavior Signature Formats

There is no standard method for sharing malware information indicating its presence on the computer system. Researchers introduce frameworks, such as Open Indicators of Compromise (OpenIOC)[185] and Malware Attribute Enumeration and Characterization (MAEC)[186], to identify malware based on its network and host level indicators instead of hash values or signatures employed in conventional security tools. Consequently, these standardized malware reporting formats provide opportunity to characterize malware samples uniformly and prevents malware community from re-analyzing samples.

5.2.1.1 Open Indicators of Compromise - OpenIOC

IOCs (Indicators of Compromise), sometimes called just indicators, are forensic artifacts of an malware attack (i.e. intrusion) that can be extracted from a host computer or network. OpenIOC is a format developed by Mandiant Corp. [187] for defining, reporting and sharing the various clues related to malware infection which enables fast and efficient malware detection schema to its users. OpenIOC supports various categories such as email, network, process, file system, registry, etc. to specify malware presence on the system. Moreover, users can use logical operators (AND and OR operator) in order to write their own complex and precise signatures that fits their environment.

While creating a signature, OpenIOC employs an XML schema which can be easily parsed and interpreted by various tools and libraries. Figure 5.2 shows an example of OpenIOC created for Stuxnet malware through Mandiant IOCe editor [188]. This signature looks for the following artifacts respectively:

- Checks if file name contains specified string.

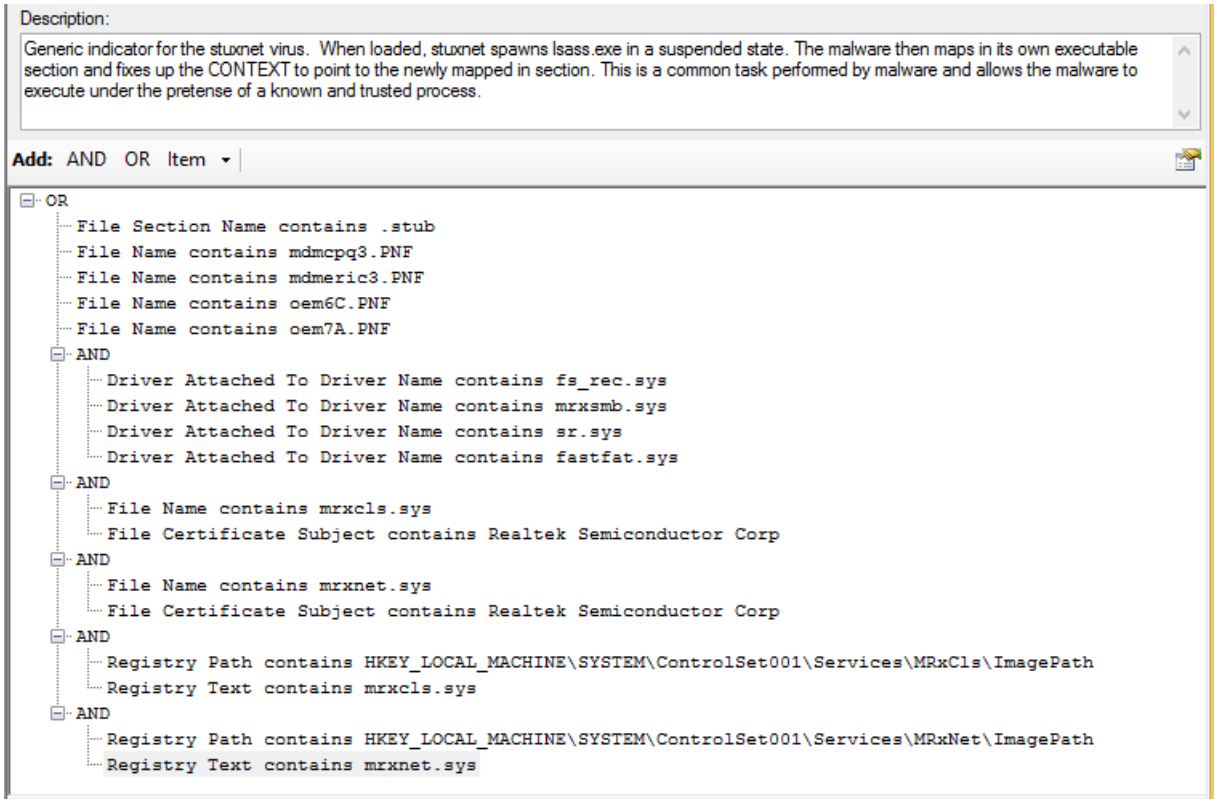


Figure 5.2: An OpenIOC format for a Stuxnet malware sample

- Searches the specified drivers; fs_rec.sys, mrxsmb.sys, sr.sys and fastfat.sys .
- Checks if there is a file named mrxcls.sys(i.e. software driver) signed by Realtek Semiconductor Corp.
- Checks if there is a file named mrxnet.sys(i.e. software driver) signed by Realtek Semiconductor Corp.
- Checks if there is a service named mrxcls on the system by enumerated the specified registry key.
- Checks if there is a service named mrxnet on the system by enumerated the specified registry key.

5.2.1.2 Malware Attribute Enumeration and Characterization - MAEC

"Malware Attribute Enumeration and Characterization (MAEC) is a standardized language developed by Mitre Corp. [189] for encoding and communicating high-fidelity information about malware based upon attributes such as behaviors, artifacts, and attack patterns" [186]. MAEC aims to eliminate the ambiguity and inaccuracy of the existing malware description schema. MAEC schemas not only reduce possible

duplication of analysis efforts, it also allows for the faster and efficient development of precaution by taking advantage of the responses to the previously observed intrusions.

MAEC language is defined by three data models including Container, Package and Bundle. Each of this models is implemented by its own XML schema. The most important information related to our study includes in MAEC Bundle section. As shown in Figure 5.3, MAEC Bundle data model consists of three interconnected layer. The key information about the layers are explained in below.

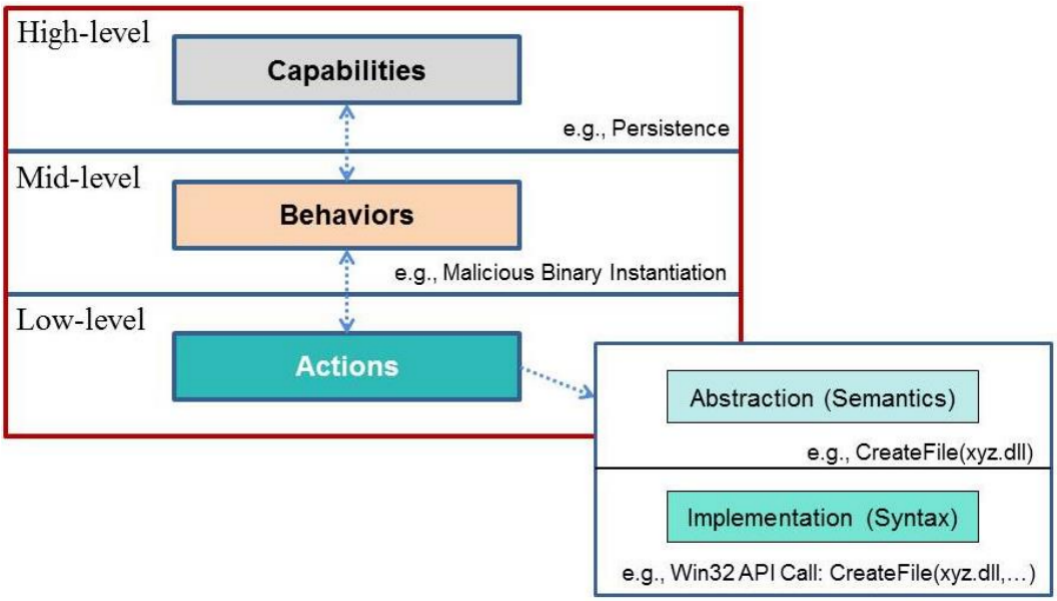


Figure 5.3: Tiers of the MAEC Bundle

- **Low Level – Actions:** At the lowest layer, MAEC try to answer question "What does the piece of malware do on a system or network?". To this end, MAEC actions characterizes all system state changes performed by malware. Although this level provides basic properties of the malware which allows us to compare malware and find similarities along the samples, it does not answer question "Why does the malware perform these actions?". Thus, these actions are needed to be interpreted by analysts to understand their goals.

- **Mid Level – Behaviors:** This tier of the MAEC defines the goals behind the low level actions. At this level, malware functionalities are extracted at significant level of abstraction in order to describe how they operate. Similar actions could be associated with different behaviors. For example, a registry key could be used to allow malware to auto-start capabilities, or it could be employed as a simple option while executing malware sample.

- **High Level – Capabilities:** At the upper-most layer, MAEC define high-level abilities possessed by a malware sample. The key difference between behavior and capability is that behavior describe how a malware operates, capability states what a malware sample is capable of doing. In this regard, a behavior may serve to describe a specific implementation of a capability which is possessed by a malware. For example, auto-starting is a behavior that is typically part of a 'Persistence' capability. Other examples of capabilities include 'Self-Defense', 'Spying', 'Data Exfiltration' and so on.

Example of MAEC Bundle Mapping

Consider the following as an example of how a malicious action can be mapped into the MAEC Bundle. Suppose the malware under consideration calls the Windows CreateService API to create a service. This event would first be mapped to the 'Create Service' Action in the lowest-layer of MAEC Bundle. After further investigation, we might conclude that the created service was used for malicious purposes such as unauthorized access to the system, thus mapping to a 'Malicious Service Installation' Behavior for middle-level. Finally, this behavior could be considered as a part of a malware 'Persistence' Capabilities. This mapping is illustrated in Figure 5.3.

5.2.2 Selected Behavioral Features

In this study, common host-based malicious features, which are also the most significant and representative features used to present malware behavior both in MAEC and OpenIOC are selected. Moreover n-gram modeling over categories of the API call and network-level intrusion alerts are employed while modeling behavior of a software. In the following, these features are given in detail.

As machine learning algorithm needs more structural features to work, feature extraction is applied on the analysis report obtained from dynamic analysis. Table 5.3 shows the selected features and their variable types used for characterizing a malware sample.

5.2.2.1 N-gram modeling over API-call Sequences

Windows API call sequence represents the behavior of executable file. As the malware authors require to call relevant API calls to achieve their malicious goals, analysis of API calls would allow malware examiner to understand the behavior of a given sample. The Windows API function calls can be grouped into various functional categories

such as system, registry, services, network activities, etc. As listed below, currently 15 different categories of API calls are used while building n-gram features.

- hooking : operations or actions to inject code
- network : high-level actions for creating network connection
- windows : actions on window objects
- process : actions on processes
- misc : various actions for additional information
- system : querying of system information
- threading : actions on thread
- synchronization : actions on synchronization objects(mutex, named pipe)
- device : actions to access real or virtual devices
- registry : actions on registry database
- filesystem : actions on file system
- services : actions on Windows services
- socket : low-level actions related to network connections

Instead of directly taking into account API call based modeling, we consider the category based modeling, which leads a lower feature space. To this end, we directly used categories of the API call. Consequently, processing of the machine learning algorithm requires less time and CPU power. Table 5.2 shows the API calls and their categories.

To specify weight of each feature obtained from n-gram extraction, the term frequency and inverse document frequency are used. **Term frequency,** or tf for short is the statistical measure for how often a term occurs in a document. Moreover, the following types of term frequency can be defined and used:

- boolean frequency for given term **t**:

$$tf(t) = \begin{cases} 1, & \text{if } \mathbf{t} \text{ occurs in a given document} \\ 0, & \text{otherwise} \end{cases}$$

- logarithmic frequency for given term **t**:

$$tf(t) = 1 + \log f(t)$$

where $f(t)$ is the number of occurances of term **t** in a given document

Table 5.2: API calls and their categories

Category	Code	API #	APIs
hooking	A	1	unhookwindowshookex
network	B	19	dnsquery_a, dnsquery_utf8, dnsquery_w, getaddrinfo, getaddrinfow, httpopenrequesta, httpopenrequestw, httpsendrequesta, httpsendrequestw, internetclosehandle, internetconnecta, internetconnectw, internetopena, internetopenw, internetopenurla, internetopenurlw, internetreadfile, internetwritefile, urldownloadtofilew
windows	C	4	findwindowa, findwindoww, findwindowexa, findwindowexw
process	D	21	createprocessinternalw, exitprocess, ntallocatevirtualmemory, ntcreateprocess, ntcreateprocessex, ntcreatesection, ntcreateuserprocess, ntfreevirtualmemory, ntopenprocess, ntopensection, ntprotectvirtualmemory, ntreadvirtualmemory, ntterminateprocess, ntwritevirtualmemory, readprocessmemory, shellexecuteexw, system, virtualfreeex, virtualprotectex, writeprocessmemory, zwmapviewofsection
misc	E	2	getcursorpos, getsystemmetrics
system	F	12	exitwindowsex, isdebuggerpresent, ldrgetdllhandle, ldrgetprocedureaddress, ldrloaddll, lookupprivilegevaluew, ntclose, ntdelayexecution, setwindowshookexa, setwindowshookexw, writeconsolea, writeconsolew
threading	G	12	createremotethread, createthread, exitthread, ntgetcontextthread, ntcreatethreadex, ntcreatethread, ntopenthread, ntresumethread, ntsetcontextthread, ntsuspendthread, ntterminatethread, rtlcreateuserthread
synchronization	H	3	ntcreatemutant, ntcreatenamedpipefile, ntopenmutant
device	I	1	deviceiocontrol
registry	J	38	ntcreatekey, ntdeletekey, ntdeletevaluekey, ntenumeratekey, ntenumeratevaluekey, ntloadkey, ntloadkey2, ntloadkeyex, ntopenkey, ntopenkeyex, ntquerykey, ntquerymultiplevaluekey, ntqueryvaluekey, ntrenamekey, ntreplacekey, ntsavekey, ntsavekeyex, ntsetvaluekey, regclosekey, regcreatekeyexa, regcreatekeyexw, regdeletekeya, regdeletekeyw, regdeletevaluea, regdeletevaluew, regenumkeyexa, regenumkeyexw, regenumkeyw, regenumvaluea, regenumvaluew, regopenkeyexa, regopenkeyexw, regqueryinfokeya, regqueryinfokeyw, regqueryvalueexa, regqueryvalueexw, regsetvalueexa, regsetvalueexw
filesystem	K	23	createdirectoryw, createdirectoryexw, removedirectorya, removedirectoryw, findfirstfileexa, findfirstfileexw, deletefilea, deletefilew, ntcreatefile, ntopenfile, ntreadfile, ntwritefile, ntdeviceiocontrolfile, ntquerydirectoryfile, ntqueryinformationfile, ntsetinformationfile, ntopendirectoryobject, ntcreatedirectoryobject, movefilewithprogressw, copyfilea, copyfilew, copyfileexw, ntdeletefile
services	L	10	controlservice, createservicea, createservicew, deleteservice, openscmanagera, openscmanagerw, openservicea, openservicew, startservicea, startservicew
socket	M	24	accept, bind, closesocket, connect, connectex, gethostbyname, ioctlsocket, listen, recv, recvfrom, select, send, sendto, setsockopt, shutdown, socket, transmitfile, wsarecv, wsarecvfrom, wsasend, wsasendto, wsasocketa, wsasocketw, wsastartup

On the other hand, to designate the global weight of the n-gram extracted from the malware dataset, **inverse document frequency**(idf) measure, an other statistical measure to indicate whether the given term is common or rare over all of the documents

(in our case document refers to malware sample). Idf is the logarithm of the total number of documents divided by the number of the documents containing the term. Given that definition, the idf give more importance to the term which rarely occurs in the document corpus.

$$idf(t) = \log \frac{total\ number\ of\ sample}{number\ of\ documents\ containing\ the\ term}$$

where the *term* occurs at least once in the document corpus.

5.2.2.2 IDS Alerts

Intrusion Detection System (IDS) is useful to inspect network stream during malware analysis for catch malicious activities over network. While IDS scanning network traffic for potential malicious activities by their rule set, they produce alerts and logs them. In our study, we used the up-to-date IDS ruleset taken from Emerging Threats [190], a leading company in the field of network-based threat detection. For modeling malware behavior it has an important option, to add IDS alerts to the feature set.

The IDS signatures consists of a header and options. The first line of the Listing 5.1 is the header of the rule, which includes basic packet header information such as IP addresses & ports, as well as actions to apply the packet like drop, alert, etc.. The other lines of the Listing 5.1 is the rule options, which includes alert message, signature patterns, rule revision, and other supported information.

After detecting network attack, IDS system produces the alert and logs it to the specified file. Typically, the IDS alert includes time, rule specific information like unique identifiers (sid and gid), revision, description of alert, alert type, priority and the IP, port and protocol type associated with the attack respectively(refer Listing 5.2). To take advantage of this useful information we parse the network alerts to make it more structure. More precisely, we only consider the definition of the alert and ignore the other remaining information. For instance, given the alert in Listing 5.2, "ET MALWARE W32/InstallRex.Adware Report CnC Beacon" is extracted and inputted as string format into online learning algorithm.

```
alert http $HOME_NET any -> $EXTERNAL_NET any (
   msg: "ET MALWARE W32/InstallRex.Adware Report CnC Beacon";
   flow:established, to_server;
   content:"POST"; http_method;
   content: "/?report_version= ";
   http_uri; content:"data=";
   http_client_body; depth:5;
   reference:md5, 9abbb5ea3f55b5182687db69af6cba66;
   classtype:trojan-activity;
```

```
    sid:2017912;
    rev:2;
)
```

Listing 5.1: An Example of IDS Rule

```
02/18/2015-08:58:12.236787  [**] [1:2017912:2]
ET MALWARE W32/InstallRex.Adware Report CnC Beacon  [**]
[Classification: A Network Trojan was detected]
[Priority: 1]
{TCP} 10.10.14.2:49324 -> 54.191.16.149:80
```

Listing 5.2: An Alert belonging to the IDS rule given in Listing 5.1

An Example for Feature Selection

As it is well-known, malware tends to utilize random names for various types of OS resources such as file, registry, etc. to make it more difficult to analyze. For this reason, run-time features are sanitized from random values which are changed by malware for each run. Consider the following as an example of how behavioral actions are mapped into the feature set. This mapping for the file with MD5=a27d774a8ce7846dfd5ae40d4411cb81 is illustrated in Table 5.3.

5.3 Online Machine Learning

The problems faced with classification of the malware using machine learning is that feature space is very large. Researchers proposes many techniques to reduce this feature space. However, their approaches do not scale well for large amount of dataset and need more computational resources. To overcome this limitation researchers propose distributed machine learning frameworks. These frameworks can be separated into two groups according to model updating mechanism: online frameworks which updates its model for each training samples and batch (or offline) frameworks updating models in regular interval. The online framework is more useful to properly separate malwares into respective classes since once a new sample appears in the wild the model needs to be updated immediately.

Online learning algorithms accommodate simple and fast model updates. Generally, they used for solving complex problems where high dimensional feature representation such as n-gram, bag-of-words demand computational and runtime efficiency (for instance natural language processing) .

Table 5.3: Features and their types

Feature Category	*Type*	*Value*
Sequnece of API category	N-gram	'ABCA', 'BACA', ...
Mutex names	String	'z3sd'
Created processes	String	'reg.exe'
Copy itself	Boolean	False
Delete itself	Boolean	False
DNS requests	String	'fewfwe.com www.hugedomains.com fewfwe.net'
Remote IPs	String	'54.209.61.132 216.38.220.26'
TCP Dst Port	String	'80'
UDP Dst Port	String	*None*
Presence of the special APIs	Boolean	'isdebuggerpresent': False, 'setwindowshook': True
Read files	String	*None*
Registry keys	String	*None*
Changed/created files	String	'% DOCUME~1%/ftpdll.dll %SYSTEM%/drivers/spools.exe %SYSTEM%/ftpdll.dll %APP_DATA%/cftmon.exe ...'
Changed/created registry keys	String	'HKCU\Software\Microsoft\Windows\ CurrentVersion\Run HKLM\SOFTWARE\Microsoft\WindowsNT\ CurrentVersion\Winlogon ...'
Downloaded EXE	String	'spools.exe cftmon.exe'
Downloaded DLLs	String	'ftpdll.dll'
User-agents	String	'_'
IDS alert	String	*None*
HTTP requests	String	'/?&v=Chatty/domain_profile.cfm? d=fewfwe&e=com'
ICMP data	String	*None*
ICMP host	String	*None*
IRC commands	String	*None*

5.3.1 Binary Classification

In general, online learning algorithm works in sequence of consecutive rounds. On each round, the algorithm takes an instance $\vec{x}_t \in \mathbb{R}^d$, d-dimensional vector, as input to make the prediction $\hat{y}_t \in \{+1, -1\}$ (for binary classification) regarding to its current prediction model. After predicting, it receives the true label $y_t \in \{+1, -1\}$ and updates its model (a.k.a hypothesis) based on prediction loss $\ell(y_t, \hat{y}_t)$ meaning the incompatibility between prediction and actual class. The goal of online learning is to minimize the total number of incorrect predictions. Pseudo-code for generic online learning is given in Algorithm-1.

Algorithm 1 Generic Binary Online Learning Algorithm

Initialize: $\vec{w}_{t=1} = (0,...,0)$
  for each round t in (1,2,..,N) **do**
    Receive instance $\vec{x}_t \in \mathbb{R}^d$
    Predict label of $\vec{x}_t$: $\hat{y}_t = sign(\vec{x}_t.\vec{w}_t)$
    Obtain true label of the $\vec{x}_t$: $y_t \in \{+1,-1\}$
    Calculate the loss: ℓ_t
    Update the weights: $\vec{w}_{t+1}$
  end for
Output: $\vec{w}_{t=N} = (w_1,...,w_d)$

5.3.2 Multi-class Classification

Similar to online binary classification, online multi-class classification operates over a sequence of training sample $(\vec{x}_1,\hat{y}_1),...,(\vec{x}_t,y_t)$. Unlike binary classification where there is only two class; $y_t \in \{+1,-1\}$, in multiclass learning there are $N - classes$; $y_t \in \mathcal{V} = \{1,...,N\}$. This makes multi-class classification algorithm more challenging to implement.

Online multiclass classification algorithms, learn from multiple classifiers, in our case a total of K classifiers are trained to obtained the model for predicting multi-class problem. The predicted label ($\hat{y}_t$ shown in Equation 5.1) is the one associated with the largest prediction value produced by the classifiers. After the prediction, the true label $y_t \in \mathcal{V}$ will be uncovered, the learner then calculates the loss function to measure the incompatibility between the prediction and the actual label. Based on the results of the loss function, the learner decides when and how to update the K classifiers at the end of each learning step. Pseudo-code for multi-class online learning is given in Algorithm-2.

$$\hat{y}_t = \arg \max_{i \in (1,...,K)} \vec{w}_{t,i}\vec{x}_t \tag{5.1}$$

In our study, i.e., multi-class classification problem:

- $\vec{x}_t$ represents feature vector of a malware instance at the t-th iteration. When implementing the algorithm for malware detection, the set of features for the given sample is constituted by using the basis of the feature vector whose feature category, its type and value is given for each independent feature in Table 5.3.
- $\vec{y}_t$ is the set of labels at the $t - th$ iteration. $\hat{y}_t$ is the output of the algorithm or more simply prediction of malware family for the given $\vec{x}_t$. More practically, in our study, the families to be matched are listed in the first column of Table 5.3, and their union constitutes the whole family set.

- ℓ_t is the function using the relation between the set of features for the given sample, the computed weight, denoted by $\vec{w}_t$ and the estimated malware label.

- $\vec{w}_{t+1}$ denotes the updated weight vector at the $(t+1) - th$ iteration towards the final prediction output.

Algorithm 2 Multi-Class Online Learning Algorithm

Initialize: $\vec{w}_{t=1} = (0, ..., 0)$
  for each round t in (1,2,..,N) **do**
    Receive instance $\vec{x}_t \in \mathbb{R}^d$
    Predict label of $\vec{x}_t$: $\hat{y}_t = \arg\max_{i \in (1,...,K)} \vec{w}_{t,i}\vec{x}_t$
    Obtain true label of the $\vec{x}_t$: $y_t \in \mathcal{V}$
    Calculate the loss: ℓ_t
    Update the weights: $\vec{w}_{t+1}$
  end for
Output: $\vec{w}_{t=N} = (w_1, ..., w_d)$

5.3.3 Online Learning Algorithms Used In This Study

Online machine learning algorithms differ according to how to initialize the weight vector $\vec{w}_{t=1}$ and update function used to alter the weight vector at the end of each round. In the following section, we discuss the basic machine learning algorithms used in our malware classification task and provide their details.

5.3.3.1 Passive-Aggressive Learning

Passive-Aggressive(PA) involves an assertive update strategy by altering the weight vector as much as needed to fulfill the constraint enforced by ongoing round. In certain learning problem which contains mislabel samples PA may drastically change its weight vectors in the wrong direction to satisfy the constraints.

PA learning is formulated as:

$$\begin{aligned} \vec{w}_{t+1} = \arg\min_{\vec{x}_t} \frac{1}{2} \left|\left| \vec{w} - \vec{w}_t \right|\right|^2 \\ subject\ to\ \ell_t(\vec{w}; (\vec{x}_t, y_t)) = 0 \end{aligned} \tag{5.2}$$

where the loss function is based on the hinge loss:

$$\ell_t(\vec{w}; (\vec{x}_t, y_t)) = \begin{cases} 0 & if\ y_t(\vec{w}.\vec{x}_t) \geq 1 \\ 1 - y_t(\vec{w}.\vec{x}_t) & otherwise \end{cases}$$

The solution to the optimization problem in Equation 5.2 has a simple closed form solution.

$$\vec{w}_{t+1} = \vec{w}_t + \tau_t y_t \vec{x}_t \qquad where \quad \tau_t = \frac{\ell_t}{\left\|\vec{x}_t\right\|^2} \tag{5.3}$$

To deal with such problems resulted from aggressive update, the two variants of PA algorithm is proposed by researchers. These algorithms introduce non-negative slack variable ξ into the optimization problem in order to obtain flexible update strategy. This slack variable can be bring into two ways: linear or quadratic form.

Considering the objective function scales linearly depending on ξ, the following constrained optimization problem is obtained.

$$\begin{aligned} \vec{w}_{t+1} &= \arg\min_{\vec{x}_t} \frac{1}{2}\left\|\vec{w} - \vec{w}_t\right\|^2 + C\xi \\ &subject\ to:\ \ell_t(\vec{w};(\vec{x}_t, y_t)) \leq \xi \quad and \quad \xi \geq 0 \end{aligned} \tag{5.4}$$

where C is a positive real number which supervises the effect of the slack variable on the objective function. In the literature, this form of the algorithm is called PA-I.

Alternatively, the objective function can be formed to scale quadratically with ξ, leading us the following constrained optimization problem,

$$\begin{aligned} \vec{w}_{t+1} &= \arg\min_{\vec{x}_t} \frac{1}{2}\left\|\vec{w} - \vec{w}_t\right\|^2 + C\xi^2 \\ &subject\ to:\ \ell_t(\vec{w};(\vec{x}_t, y_t)) \leq \xi \end{aligned} \tag{5.5}$$

where C is again a positive real number which supervises the effect of the slack variable on the objective function. This obtained algorithm is termed as PA-II.

PA-I and PA-II has simple closed-form solution,

$$\begin{aligned} &\vec{w}_{t+1} = \vec{w}_t + \tau_t y_t \vec{x}_t \\ &\text{where} \quad \tau_t = \min\left\{C, \frac{\ell_t}{\left\|\vec{x}_t\right\|^2}\right\} \text{ (for PA-I)} \\ &\tau_t = \frac{\ell_t}{\left\|\vec{x}_t\right\|^2 + \frac{1}{2C}} \text{ (for PA-II)} \end{aligned} \tag{5.6}$$

5.3.3.2 Confidence-Weighted Learning

Dredze et al. introduce confidence-weighted(CW) learning for binary classification problem. CW employs a distribution function to update weight vector instead of using single vector like PA algorithm. CW algorithm maintains a Gaussian distribution of

weights: $\mathcal{N}(\vec{\mu}, \Sigma)$ where $\vec{\mu} \in \mathbb{R}^d$ mean vector and $\Sigma \in \mathbb{R}^{d \times d}$ covariance matrix. Given an input instance $x_t \in \mathbb{R}^d$ Gibbs classifier calculates the weight vector $\vec{w}$ from the Gaussian distribution and makes a prediction $\{+1, -1\}$ according to $sign(\vec{x_t} \cdot \vec{w})$ function.

CW learning algorithm updates the weight distribution by minimizing the Kullback-Leibler diverge between the new weight distribution and the previous one while ensuring that the probability of correct prediction is no smaller than a given threshold value(the confidence).

$$\begin{aligned} (\vec{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\vec{m}_t, \Sigma} D_{KL}(\mathcal{N}(\vec{\mu}, \Sigma), \mathcal{N}(\vec{\mu}_t, \Sigma_t)) \\ \text{subject to:} \quad Pr_{w \sim \mathcal{N}(\vec{\mu}, \Sigma)}[y_t(\vec{w} \cdot \vec{x}_t) \geq 0] \geq \eta \end{aligned} \tag{5.7}$$

where η is threshold value named as confidence parameter. Dredze et al. proved that this optimization problem can be solved in closed-form [191]:

$$\begin{aligned} \vec{\mu}_{t+1} &= \vec{\mu}_i + \alpha_i \Sigma_i \vec{x}_i \\ \Sigma_{t_+1} &= \Sigma_t - \beta_t \Sigma_t \vec{x}_t^T \vec{x}_t \Sigma_t \end{aligned} \tag{5.8}$$

where updating coefficients are computed as follows:

$$\alpha_t = \max\left\{0, \frac{1}{v_t \zeta}\left(-m_t \psi + \sqrt{{m_t}^2 \frac{\phi^4}{4} + v_t \phi^2 \zeta}\right)\right\}$$

$$\beta_t = \frac{\alpha_t \phi}{\sqrt{u_t} + v_t \alpha_t \phi}$$

where $u_t = \frac{1}{4}(-\alpha_t v_t \phi + \sqrt{{\alpha_t}^2 {v_t}^2 \phi^2 + 4 v_t})^2$, $v_t = \vec{x}_t^T \Sigma_t \vec{x}_t$, $m_t = y_t(\vec{\mu}_t \cdot \vec{x}_t)$, $\phi = \Phi^-1(\eta)$ (Φ is the cumulative function of the normal distribution), $\psi = 1 + \frac{\phi^2}{2}$ and $\zeta = 1 + \phi^2$.

5.3.3.3 Adaptive Regularization of Weights

As CW learning, Adaptive Regularization of Weights(AROW) learning assumes a Gaussian distribution over weight vectors with mean vector $\vec{\mu} \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$. However, unlike CW learning, AROW employs adaptive update method while handling a new sample at each learning step. This makes AROW more resistant to instantaneous changes when encounter mislabeled training sample in learning phase. Particularly, Crammer et al. summarized and formulated AROW learning by the following optimization problem:

$$(\vec{\mu}_{t+1}, \Sigma_{t+1}) = \arg\min_{\vec{m}_t, \Sigma} D_{KL}(\mathcal{N}(\vec{\mu}, \Sigma), \mathcal{N}(\vec{\mu}_t, \Sigma_t)) + \frac{1}{2\gamma}{}^2 \ell^2(\vec{\mu}; (\vec{x}_t, y_t)) + \frac{1}{2\gamma}\vec{x}_t^T \Sigma_t \vec{x}_t \tag{5.9}$$

where $\ell^2(\vec{\mu}; (\vec{x}_t, y_t)) = (\max\{0, 1 - y_t(\vec{\mu} \cdot \vec{x}_t)\})^2$ and γ is a regularization parameter. The optimization of AROW learning has a closed-form similar to CW, but has different coefficients:

$$\alpha_t = \ell(\vec{\mu}_t; (\vec{x}_t, y_t))\beta_t$$
$$\beta_t = \frac{1}{\vec{x}_t^T \Sigma_t \vec{x}_t + \gamma}$$

5.3.3.4 Gaussian Herding

Gaussian Herding(NHERD) is a modified version of PA-II which has quadratic function for updating weight. NHERD assumes a Gaussian distribution over weight vectors with mean vector $\vec{\mu} \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d \times d}$ like AROW and CW. Furthermore, on round t, NHERD employs a liner transformation of the weight distributions with matrices A_t. In particular, Crammer and Lee formulate the loss function of NHERD as follows [192]:

$$(\vec{\mu}_{t+1}, A_t) = \arg\min_{\vec{m}, A} \frac{1}{2}(\vec{\mu} - \vec{\mu}_t)^T \Sigma_t^{-1}(\vec{\mu} - \vec{\mu}_t) + \frac{1}{2} Tr\left((A - I)^T \Sigma_t^{-1}(A - I)\Sigma_t\right) + C(1 - \gamma_t)^2 + \frac{C}{2}\vec{x}_t^T A \Sigma_t A^T \vec{x}_t \tag{5.10}$$

The objective function of NHERD has a closed-form similar with the following coefficients:

$$\alpha_t = \frac{(1 - \gamma_t)}{v_t + \frac{1}{C}}$$
$$\beta_t = 2C + C^2 v_t$$

where $\gamma_t = y_y(\vec{\mu}_t \cdot \vec{x}_t), v_t = \vec{x}_t^T \Sigma_t \vec{x}_t$ and C is a aggressiveness parameter (threshold value).

5.4 Jubatus Online Learning Framework

In this work, Jubatus, an online machine learning framework is preferred to classify malware samples based on their behavioral patterns. When compared to current existing online learning tools LIBOL [193] and MOA (Massive Online Analysis) [194], Jubatus is the only platform supporting parallel computing with distributed computers. Therefore, we deployed distributed jubatus servers in order to handle malware classification task efficiently and consistently. Now, let's look at the key features of Jubatus framework and its configuration.

5.4.1 Jubatus Architecture

Essentially, Jubatus aims to combine machine learning and parallel computing together. Besides that, Jubatus supports various online learning algorithm in different categories such as classification, regression, clustering, graph matching, etc. and has real-time processing and scale-up abilities. All these features make Jubatus to be a versatile and powerful tool for mining large-scale data.

Jubatus employs a client-server architecture to control components and to share computationally expensive processes to its distributed servers over the network. The task can be run either on single server or on multiple servers if you need scalability. Jubatus framework uses a special method called loose model sharing which includes the three major operations:

- **UPDATE:** Update operation corresponds to the training phase of online learning algorithm. Initially, each server has a local model which is updated once a new sample is submitted. In this step, a training sample will automatically be sent to randomly selected server or to specified servers based on the user preferences. After processing the sample, the corresponding server updates its local model in an online learning manner.

- **MIX:** The MIX operation is the key step corresponding the sharing of the local models of each server in order to build final version of the model. At the beginning of each MIX operation, a server is randomly selected as a parent node for the remaining servers. First of all, the parent node connects the rest of the servers to gather their local model built in the previous step. Then the parent node merges each of the local models into one model and pushes up-to-date model to the others. At the end of MIX, all servers have the same model.

- **ANALYZE:** This step is responsible of prediction. In this step, a given test sample is also sent to one server. Then, according to its model the server makes a

prediction for the sample.

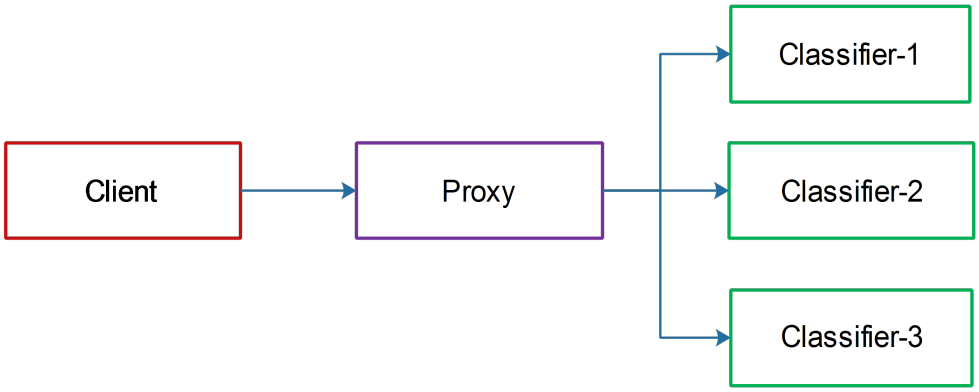


Figure 5.4: Distributed Mode Jubatus

Jubatus composed of the following components for distributed mode. Figure 5.4 illustrates the interaction between each of these nodes.

- **Jubatas client:** A Jubatus client is a component which requests a task from Jubatus keeper to execute and obtains the result from the keeper. Although one client is enough for running task on Jubatus, sometimes when training data is huge, it is a good practice to use multiple Jubatus clients .
- **Jubatus keeper (proxy):** The Jubatus keeper, is the important component which is responsible controlling Jubatus nodes to accomplish scalable distributed computing environment. More specifically, it requests a task to Jubatus nodes and receives the results from them.
- **Jubutus node (server):** A Jubatus node is a component where a specified machine learning algorithm is executed. The examples of such algorithms are recommendation, classification, clustering, regression, etc..

5.4.2 Data Conversion Engine

The unstructured data such as texts or multi-media content can not be used directly in machine learning. Instead, this data are converted to "feature vectors". Jubatus uses data-conversion engine to simplify feature extraction process by employing a simple configuration file. In this file, features are defined in terms of key-value pairs, called "datum". Jubatus has three default datums as follows:

1. string_values, whose key and value are both string,
2. num_values, whose key is string, but value is numeric data,
3. binary_values, whose key is string, but value is arbitrary binary data.

5.4.3 Our Jubatus Deployment

We set up a three machine cluster for Jubatus, allows us not only to speed the classification process but also to handle large dataset. While analysis, each machine done its jobs and deliver the result to the manager of the cluster. Since Jubatus performs its action on memory to respond rapidly, it needs more memory. Accordingly, we deployed each servers with 32 GB memory.

Listing 5.3 shows an example configuration used to extract features from the behavioral reports of the analyzed file. Based on the configuration file, each word separated by spaces is used as a feature for string data. For numeric data, each value is used itself as a feature. Specifically, for the "category_list", features extracted by N-gram analysis is employed. This configuration file and its parameters directly and significantly affects the accuracy of the model. We will discuss the affects of these parameters in the following chapter (Chapter 6).

```
{
  "method": "CW",
  "converter": {
    "num_filter_types": {},
    "num_filter_rules": [],
    "string_filter_types": {},
    "string_filter_rules": [],

    "binary_types": {},
    "binary_rules": [],

    "num_types": {},
    "num_rules": [
      { "key": "*", "type": "num"}
    ],
    "string_types": {
      "5gram": {"method": "ngram", "char_num": "5"}
    },
    "string_rules": [

      { "key": "category_list", "type": "5gram", "sample_weight": "tf", "global_weight": "idf"
      },

      { "key": "*", "except": "category_list", "type": "space", "sample_weight": "bin", "
    global_weight": "bin" }

    ]
  },
  "parameter": {
    "regularization_weight" : 1.0
  }
}
```

Listing 5.3: An example for the configuration of data conversion

After installing and executing Jubaclassifiers on the remote servers(refer to Section A.3 for further details), we start to evaluate the proposed malware classification method. The flow of classification includes the following tasks. The Python pseudo-code for these tasks is given in Listing 5.4.

- Connect to Jubaclassifier
- Prepare the training and testing data
- Create model from training data
- Classify the testing data
- Evaluate the result

```
#!/usr/bin/env python
import jubatus
from jubatus.common.datum import Datum
from jubatus.classifier import client

host = '127.0.0.1'
port = 9199
name = 'test'
timeout = 1000
# step 1: Connect to Jubaclassifier
client = jubatus.Classifier(host, port, name, timeout)

# step 2: Prepare the training and testing data
training_set = [some samples for training]
testing_set = [some samples for testing]

# step 3: Create the model
client.train(training_set)

# step 4: Classify testing set
results = client.classify(testing_set)

# step 5: Evaluate the results
for result in results:
  print testing_label[i][o] + "," + max(result, key=lambda x: x.score).
   label

# optional : clear the model
client.clear()
```

Listing 5.4: Python pseudo-code for the classification task

5.5 Conclusion

As it is well-known fact that most of the current malware samples are derived from existing ones, and, if these samples are armed with obfuscation techniques common security solutions can be easily evaded. In this chapter, we propose a novel approach for classifying malicious programs efficiently by using runtime artifacts while being robust to obfuscation. The presented dynamic malware analysis setup is usable on large scale in real world. We propose a malware classification method by using online machine learning algorithm. The proposed method employs run-time behaviors of an executable to build feature vector. All details of the proposed approach is provided throughout this chapter. In this chapter, the following key points of the classification system were explained:

- Behavioral feature vector for representing a software
- Running configuration of the Jubatus data conservation
- Pseudo-code for classification task

In the following chapter, we evaluate the proposed classification schema and discuss the contributions of our thesis in the context of other existing work.

Chapter 6

Evaluation

This chapter evaluates the proposed approach to classify malware samples according to their behavioral profiles obtained dynamic analysis.

In this section, we present the conducted experiment to evaluate the proposed malware classification approach. Firstly the used malware dataset is described in detail. Then, evaluation measures are explained and in conclusion the obtained results are provided and discussed.

6.1 The Malware Dataset

The testing malware dataset is obtained from "Virusshare Malware Sharing" platform [195] which provides huge amount of malware in different types including PE, HTML, Flash, Java, PDF etc. As VirMon only analyze executable files, we select only executable file. To understand new malware tends, the samples shared by Virusshare in the first quarter of the 2014 are chosen for behavioral analysis.

All the experiments were conducted under the Windows XP SP 3 operating system with Intel(R) Core(TM) i5-2410M@2.30 GHz processor and 1 GB of RAM. The analysis with 50 guest machines takes 15 days to analyze approximately 80.000 samples. In other words, the average time required to analyze a file is approximately one minute.

However, some files did not run because they require more hardware specification or need newer .NET version than the one located in Windows XP SP3. Besides, sometimes the analysis procedure failed. At the end of the analysis, 60.000 files are correctly analyzed and reported. However, 41% samples does not illustrate enough activities. Since these samples can cause false positives, we removed them from the dataset. Figure 6.1 shows the distribution of the dynamic malware analysis results

about the evaluation set. IDS alerts captured by VirMon for the succeeded analysis is given in Table 6.1. Moreover, Figure 6.2 shows the distribution of the successfully analyzed malware sample according to first scan time in Virustotal.

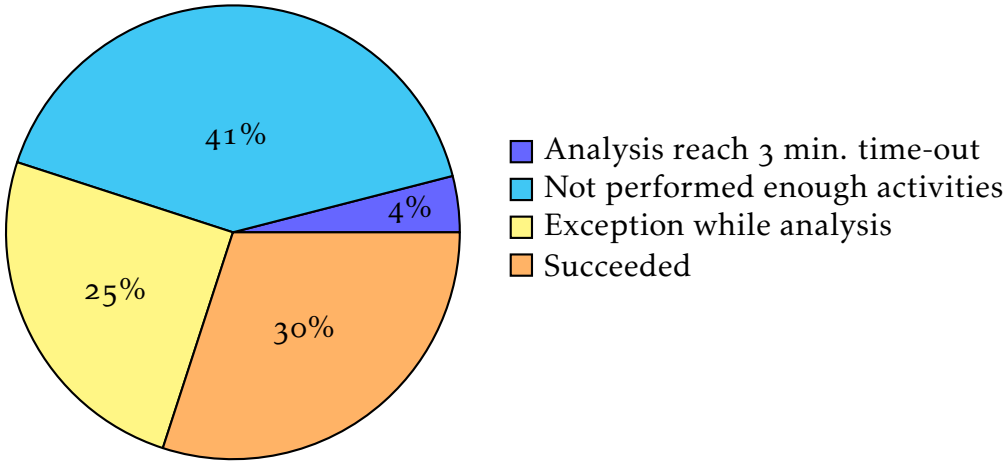


Figure 6.1: Results of dynamic analysis about the evaluation set

Table 6.1: Categories of the IDS signature extracted from dynamic analysis

Signature categories	Number of sample
A Network Trojan was detected	8041
Potential Corporate Privacy Violation	836
Misc activity [1]	551
Potentially Bad Traffic	139
Attempted User Privilege Gain	20
Attempted Information Leak	16

[1] Various kinds of attacks

For labeling malware samples, Virustotal, an online web-based multi anti-virus scanner, is used. Although Virustotal provides public API which allows the user to automate various task e.g. the scanning the suspicious file or searching scan result of a given hash value, users can make at most 4 requests in any given 1 minute time frame. However, curiously enough, Virustotal does not take any precaution like rate quota (or rate limiting) to limit the request directly targeting its web site. By taking advantage of this, we wrote a simple python script to access the scan results of the files by searching their sha256 values. The pseudo-code of this script is given in Listing 6.1. Interested users can refer to [10] for the source-code of the script.

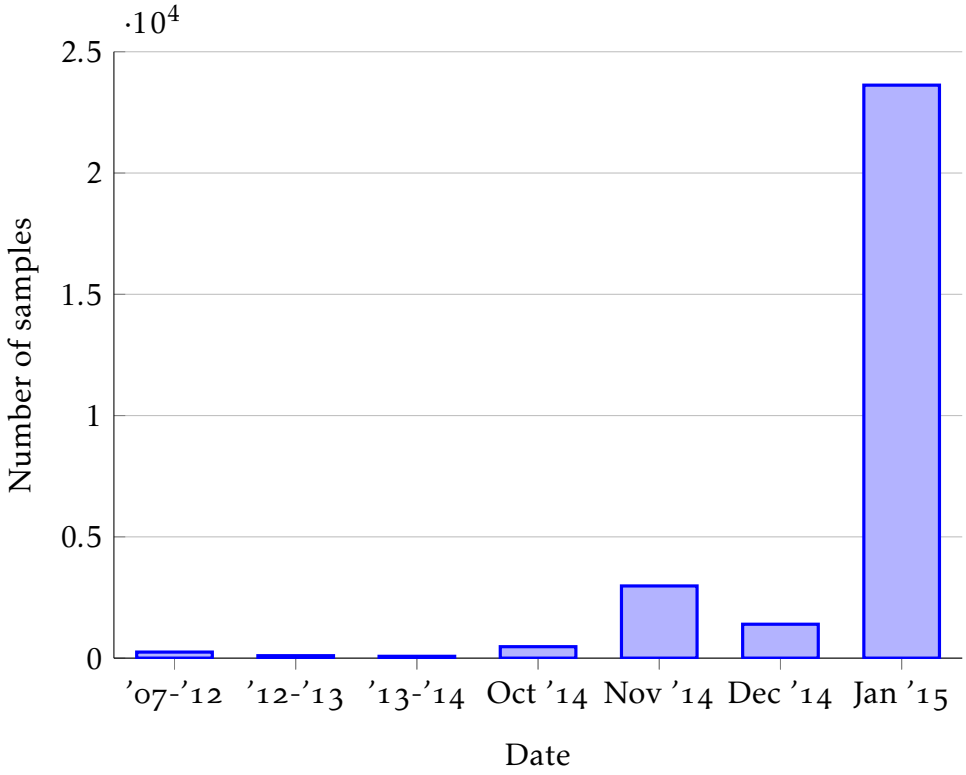


Figure 6.2: Distribution of the malware dataset according to first scan time in Virustotal[105]

```
# create http headers conscientiously otherwise virustotal does not tackle
    the request
headers = {
  'Accept-Language': 'en-US',
  'Accept-Encoding': 'gzip, deflate',
  'Connection': 'Keep-Alive',
  'Accept': 'text/html, application/xhtml+xml, */*',
  'user-agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0)
    like Gecko',
  'Host': 'www.virustotal.com'
  }

# sha256 value of the file to search
sha256 = "sha256 value of the file"
# create the URL for the specified file
url = "https://www.virustotal.com/en/file/{}/analysis/"
url = url.format(sha256)
response = requests.get(url, headers=headers)
# Parse the response in order to obtain AV-scan results and other useful
    information (such as first scan date)
```

Listing 6.1: Python pseudo-code for the searching scan result in Virustotal

As it is well known fact that malware labeling differs between different anti-virus engines [196, 197]. To be more consistent in labeling, the most common scan result is determined as tag of a sample. As the labeling process completely affects the classification accuracy, researchers need to be attentive to anti-virus labeling and sometimes cross check the labeling results. Table 6.2 shows the distribution of the malware classes used to evaluate the proposed method.

6.2 Performance Measures

To evaluate the proposed method class-specific measures like ***precision***, ***recall*** (a.k.a. sensitivity), ***specificity***, ***balanced accuracy*** and ***overall accuracy*** (the overall correctness of the model) are used.

Recall is the probability for a sample in class c to be classified correctly, the maximum value 1 means that the classifier is always correct when it estimates whether an instance belongs to class c. On the contrary, specificity is the probability for a sample not in class c to be classified correctly, the maximum value 1 means that the classifier is always correct when it estimates that an instance does not belong to class c. Moreover, precision gives probability for an estimated instance as class c to be actually in class c. Low precision means that a large number of samples were incorrectly classified as belonging to class c.

Balanced accuracy is another class-specific performance indicator used to handle misleading overall accuracy when the dataset is imbalanced. More specifically, it assesses the errors for each class and is calculated as the arithmetic mean of the specificity and recall. Whereas, overall accuracy is calculated as the sum of correct classifications divided by the total number of samples. The performance measures can be listed as follows:

$$precision = \frac{tp}{tp + fp} \tag{6.1}$$

$$recall = \frac{tp}{tp + fn} \tag{6.2}$$

$$specificity = \frac{tn}{tn + fp} \tag{6.3}$$

$$\begin{aligned} balanced\ accuracy &= \frac{recall + specificity}{2} \\ &= \frac{1}{2}\left(\frac{tp}{tp + fn} + \frac{tn}{tn + fp}\right) \end{aligned} \tag{6.4}$$

Table 6.2: Malware families and class-specific performance measures

Family	*Code*	*#*	*Precision*	*Recall*	*Specificity*	*Balanced Accuracy*
AdGazelle	ADG	121	0.59	1.00	0.99	1.00
Adw.ScreenBlaze	ASC	40	1.00	1.00	1.00	1.00
Adware.Agent.NZS	AAN	28	1.00	1.00	1.00	1.00
Adware.BetterSurf	ABE	39	1.00	1.00	1.00	1.00
Adware.Bprotector	ABP	91	1.00	1.00	1.00	1.00
Aliser	ALI	48	0.71	1.00	1.00	1.00
Almanahe.D	ALD	16	1.00	1.00	1.00	1.00
Amonetize	AMO	67	1.00	0.29	1.00	0.64
Backdoor.Fynloski.C	BFC	42	1.00	1.00	1.00	1.00
Backdoor.SpyBot.DMW	BSD	20	1.00	1.00	1.00	1.00
Banker	BAN	49	0.83	1.00	1.00	1.00
Barys	BAR	403	0.40	0.50	0.98	0.74
Bundler.Somoto	BSO	982	1.00	1.00	1.00	1.00
Chinky	CHI	433	1.00	0.30	1.00	0.65
Conjar	CON	59	0.45	0.83	1.00	0.91
Dialer.Adultbrowser	DAD	25	1.00	0.33	1.00	0.67
FakeAlert	FAL	25	1.00	0.33	1.00	0.67
FakeAV	FAV	38	0.43	1.00	1.00	1.00
Gael	GAE	10	1.00	1.00	1.00	1.00
Hotbar	HOB	126	0.63	1.00	1.00	1.00
Jeefo	JEE	49	0.83	1.00	1.00	1.00
Kates	KAT	18	0.00	0.00	0.98	0.49
Keylog	KEL	21	1.00	1.00	1.00	1.00
Parite	PAR	370	0.97	1.00	1.00	1.00
PoisonIvy	POI	38	1.00	1.00	1.00	1.00
Renos	REN	116	0.75	0.55	1.00	0.77
Sality	SAL	604	0.91	0.98	1.00	0.99
Sirefef	SIR	51	0.22	1.00	0.99	0.99
Skintrim	SKI	202	1.00	0.55	1.00	0.78
SMSHoax	SMH	43	0.67	1.00	1.00	1.00
Swizzor	SWI	536	1.00	0.96	1.00	0.98
Trojan.Agent.VB	TAV	197	0.93	0.70	1.00	0.85
Trojan.Clicker.MWU	TCM	17	1.00	1.00	1.00	1.00
Trojan.Crypt	TCR	58	0.86	1.00	1.00	1.00
Trojan.Downloader.FakeAV	TDF	16	1.00	0.33	1.00	0.67
Trojan.Generic.1733394	TG1	2507	1.00	1.00	1.00	1.00
Trojan.Keylogger.MWQ	TKM	21	1.00	1.00	1.00	1.00
Trojan.Patched.HE	TPH	28	1.00	1.00	1.00	1.00
Trojan.Startpage.ZQR	TSZ	23	1.00	0.33	1.00	0.67
Trojan.Stpage	TSP	136	1.00	1.00	1.00	1.00
Trojan.VB.Bugsban.A	TVB	199	1.00	0.15	1.00	0.58
Variant.Application.Yek	VAY	18	1.00	1.00	1.00	1.00
Virtob	VIT	763	0.69	0.48	0.99	0.74
VJadtre	VJA	134	1.00	0.69	1.00	0.85
Win32.Valhalla.204	WV2	20	1.00	1.00	1.00	1.00
Win32.Viking.AU	WVA	77	0.67	1.00	1.00	1.00
Worm.AutoIt	WAA	53	0.06	1.00	0.94	0.97
Worm.Generic.384701	WG3	7231	1.00	1.00	1.00	1.00
Worm.Hybris.PLI	WHP	998	1.00	0.01	1.00	0.50
Worm.P2P.Palevo	WPP	24	1.00	1.00	1.00	1.00
Zusy	ZUS	670	0.84	0.93	0.99	0.96

$$accuracy = \frac{correctly\ classified\ instances}{total\ number\ of\ instances} \tag{6.5}$$

Based on a particular class c;

- True positives (tp) refer to the number of the samples in class c that were correctly classified.
- True negatives (tn) are the number of the samples not in class c that were correctly classified.
- False positives (fp) refer the number of the samples not class in c that were incorrectly classified.
- False negatives (fn) are the number of the samples in class c that were incorrectly classified.

The terms positive and negative indicate the classifier's prediction, and the terms true and false indicate whether that prediction matches with ground truth label.

6.3 Results

The following online classification algorithms are used in our distributed computing environment in order to empirically obtain the best accuracy.

- Passive-Aggressive I (PA-I) [198]
- Passive-Aggressive II (PA-II) [198]
- Confidence Weighted Learning (CW) [191]
- Adaptive Regularization of Weight Vectors (AROW) [199]
- Normal Herd (NHERD) [192]

6.3.1 Parameter Tuning

Setting appropriate parameters plays a key role in determining the accuracy of the online learning algorithms. Some online learning algorithms have no parameter (e.g. perceptron) and some have multiple parameters, but the algorithms evaluated in our study have a common parameter named regularization weight and donated by C. Regularization weight is the parameter to control the sensitivity of the learning. Indeed, the bigger it is, the faster the algorithm can build the model. However, in this case the constructed model becomes more sensitive to the noise data. In other

word, the regularization weight is the parameter typically used to trade off between the model build time and the sensitivity of the model.

Furthermore, we adjust the importance of the elements in feature vector to obtain maximum classification accuracy. To this end, as indicated in Listing 5.3, the weight of each string-type feature vector is calculated with the value of **sample weight** and **global weight**. Indeed, the weight of each feature is the product of these two values. The "sample weight" specifies the weight of each feature without considering other inputted data, whereas the "global weight" specifies the overall weight which is calculated from the data inputted so far. For the number type features, we used given number as its weight. The feature types and their sample and global weight values are given in Table 6.3.

Table 6.3: The weights of each features types and their meanings

Feature	**Sample Value - Meaning**	**Global Value - Meaning**
N-gram over API categories	**tf:** frequency of the feature in given string	**idf:** the inverse of logarithm of normalized document frequency
String Type Features except n-gram	**bin:** 1 for all features and all data.	**bin:** 1 for all features and all data.
Number Type Features	**num:** use given number itself as weight	Not applicable

To clarify the difference of the sample and global weight consider the following example. 'ABCDE' n-gram feature is appeared 60 times in a malware sample. Beside that, 'ABCDE' feature is included in 500 malware samples of all 20000 malware samples. According to the given scenario, the sample weight of the 'ABCDE' is 60 and global weight is $idf = log(20000/500)$. As a result of the two parameters, the overall weight of the 'ABCDE' feature is $log(20000/500) \times 60$.

All classifiers were evaluated by utilizing 10-fold cross-validation, the initial malware set are randomly divided into 10 subset each of them is approximately equal size and training and testing is performed 10 times. The following tables shows training and testing accuracy of the selected classifiers based on various N-gram and regularization weight (C), a parameter employed when updating objective function (model). The bigger regularization weight is, the faster model is created. However, the created model becomes more dependent to training set and more susceptible to noise data.

The most accurate classification results for training and testing are obtained through CW algorithm when regularization weight equals to 4.0 and N equals to 6. The training and testing accuracies of the each algorithm are given in the following

tables. From these tables, it is easily noticeable that the classification accuracy generally diminishes as C increases. Besides that, when N increased until a threshold, the exacted features become more representative and distinct for malware families. However, above this threshold, the feature set becomes unique for each malware sample which causes the classification accuracy to drastically decrease.

	Regularization Weight(C)													
	C=1.0		C=2.0		C=3.0		C=4.0		C=5.0		C=10.0		C=100.0	
N	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
3	0.890	0.886	0.893	0.884	0.800	0.828	0.690	0.768	0.617	0.616	0.340	0.33	0.083	0.186
4	0.907	0.894	0.920	0.898	0.910	0.900	0.917	0.894	0.913	0.896	0.883	0.874	0.733	0.632
5	0.907	0.900	0.913	0.902	0.923	0.904	0.930	0.906	0.927	0.904	0.930	0.906	0.877	0.816
6	0.917	0.896	0.920	0.908	0.927	0.908	0.940	0.918	0.937	0.908	0.937	0.904	0.883	0.826
7	0.917	0.900	0.930	0.910	0.930	0.912	0.930	0.910	0.930	0.910	0.933	0.906	0.500	0.824
8	0.910	0.900	0.923	0.904	0.933	0.908	0.933	0.908	0.930	0.904	0.937	0.908	0.853	0.866
9	0.913	0.898	0.923	0.902	0.923	0.902	0.920	0.904	0.927	0.906	0.930	0.900	0.907	0.882
10	0.913	0.718	0.930	0.898	0.930	0.902	0.920	0.898	0.927	0.904	0.920	0.898	0.913	0.882

Table 6.4: Training & testing accuracy of CW

	Regularization Weight(C)													
	C=1.0		C=2.0		C=3.0		C=4.0		C=5.0		C=10.0		C=100.0	
N	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
3	0.557	0.582	0.707	0.622	0.700	0.634	0.703	0.634	0.700	0.618	0.700	0.532	0.677	0.644
4	0.813	0.814	0.807	0.808	0.793	0.798	0.800	0.802	0.800	0.796	0.807	0.796	0.807	0.796
5	0.797	0.628	0.797	0.780	0.783	0.782	0.773	0.772	0.783	0.790	0.770	0.774	0.777	0.782
6	0.833	0.838	0.830	0.828	0.827	0.828	0.817	0.818	0.820	0.824	0.813	0.830	0.810	0.816
7	0.833	0.830	0.830	0.820	0.827	0.822	0.820	0.828	0.827	0.822	0.837	0.826	0.833	0.818
8	0.840	0.830	0.837	0.828	0.840	0.828	0.827	0.820	0.823	0.824	0.827	0.814	0.827	0.826
9	0.837	0.828	0.847	0.828	0.843	0.824	0.847	0.824	0.820	0.816	0.837	0.824	0.827	0.816
10	0.830	0.828	0.827	0.820	0.830	0.812	0.820	0.808	0.827	0.810	0.817	0.806	0.807	0.810

Table 6.5: Training & testing accuracy of AROW

	Regularization Weight(C)													
	C=1.0		C=2.0		C=3.0		C=4.0		C=5.0		C=10.0		C=100.0	
N	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
3	0.247	0.130	0.103	0.164	0.110	0.080	0.237	0.154	0.163	0.112	0.147	0.142	0.060	0.104
4	0.417	0.322	0.460	0.328	0.477	0.344	0.483	0.322	0.333	0.356	0.493	0.348	0.390	0.400
5	0.567	0.432	0.570	0.462	0.577	0.458	0.547	0.434	0.567	0.450	0.523	0.438	0.510	0.330
6	0.490	0.386	0.463	0.368	0.473	0.364	0.443	0.342	0.440	0.336	0.447	0.346	0.427	0.330
7	0.423	0.362	0.383	0.312	0.350	0.328	0.327	0.276	0.333	0.288	0.317	0.222	0.310	0.266
8	0.347	0.268	0.300	0.234	0.260	0.214	0.243	0.222	0.240	0.204	0.230	0.220	0.207	0.176
9	0.317	0.206	0.270	0.192	0.277	0.146	0.160	0.132	0.127	0.160	0.107	0.124	0.107	0.142
10	0.360	0.260	0.317	0.192	0.180	0.216	0.160	0.132	0.140	0.168	0.120	0.136	0.103	0.118

Table 6.6: Training & testing accuracy of NHERD

The class-wise results for the most successful algorithm(i.e. CW) with the most appropriate parameters (C=4.0 and N=6) are given in Table 6.2. These results indicate that perfect precision and recall value (*i.e.*, 1.0) is assured for 20 out of 51 families. For example, Adw.ScreenBlaze, Worm.Generic.384701 and Gael is one of these families. Accordingly, the classifier estimates them without any error. Worm.Hybris.PLI family

	Regularization Weight(C)													
	C=1.0		C=2.0		C=3.0		C=4.0		C=5.0		C=10.0		C=100.0	
N	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
3	0.373	0.406	0.370	0.370	0.377	0.374	0.377	0.346	0.377	0.364	0.367	0.460	0.370	0.448
4	0.763	0.786	0.763	0.768	0.770	0.784	0.773	0.770	0.763	0.774	0.770	0.774	0.770	0.776
5	0.747	0.752	0.733	0.740	0.743	0.760	0.747	0.744	0.737	0.748	0.740	0.750	0.743	0.754
6	0.787	0.776	0.787	0.776	0.783	0.788	0.783	0.780	0.797	0.776	0.777	0.774	0.787	0.772
7	0.807	0.780	0.783	0.786	0.800	0.792	0.793	0.788	0.797	0.778	0.793	0.792	0.797	0.786
8	0.793	0.788	0.807	0.786	0.800	0.776	0.800	0.802	0.797	0.790	0.803	0.790	0.803	0.792
9	0.797	0.780	0.797	0.800	0.803	0.776	0.810	0.788	0.800	0.778	0.793	0.774	0.800	0.778
10	0.777	0.760	0.780	0.776	0.787	0.764	0.783	0.770	0.777	0.760	0.783	0.766	0.777	0.762

Table 6.7: Training & testing accuracy of PA-I

	Regularization Weight(C)													
	C=1.0		C=2.0		C=3.0		C=4.0		C=5.0		C=10.0		C=100.0	
N	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
3	0.370	0.358	0.357	0.422	0.353	0.372	0.340	0.400	0.350	0.376	0.373	0.354	0.370	0.352
4	0.777	0.768	0.773	0.616	0.777	0.782	0.767	0.772	0.780	0.768	0.770	0.784	0.770	0.776
5	0.750	0.756	0.737	0.754	0.740	0.760	0.737	0.748	0.743	0.748	0.743	0.746	0.737	0.748
6	0.783	0.784	0.780	0.776	0.783	0.778	0.783	0.778	0.790	0.772	0.787	0.782	0.783	0.784
7	0.797	0.796	0.800	0.788	0.797	0.790	0.803	0.782	0.797	0.782	0.793	0.800	0.797	0.784
8	0.800	0.780	0.810	0.782	0.537	0.788	0.797	0.790	0.803	0.808	0.803	0.794	0.807	0.788
9	0.800	0.786	0.797	0.756	0.783	0.774	0.793	0.780	0.803	0.778	0.797	0.768	0.803	0.762
10	0.783	0.766	0.793	0.772	0.783	0.782	0.787	0.778	0.793	0.768	0.777	0.762	0.780	0.766

Table 6.8: Training & testing accuracy of PA-II

exhibits perfect precision but low recall, indicating the classifier inaccurately estimates almost all instances as being to other families. For Kates family, the classifier produces both zero precision and recall, which indicates that it never correctly classified an instance belonging to Kates family, in other words *tp* is 0. It is important to note that the classifier estimates the Worm.Generic.384701 which has the highest number of samples and covers almost 40% of dataset without any error.

To analyze how well the CW classifier can recognize instance of different classes, we also created confusion matrix as shown in Figure 6.3. The confusion matrix displays the number of correct and incorrect predictions made by the classifier with respect to ground truth (actual classes). The matrix contains $n \times n$ entries, where n is the number of classes. The rows of the table correspond to actual classes and columns correspond to predicted classes. The diagonal elements in the matrix represent the number of correctly classified instances for each class, while the off-diagonal elements represent the number of misclassified elements by the classifier. The higher the diagonal values of the confusion matrix are, the better the model fits the dataset (high accuracy in individual family predictions).

From the confusion matrix, it can be seen that Worm.Hybris.PLI and Virtob wrongly estimated as Worm.Autoit and Kates, respectively. A quick research on the Internet shows us that some AV vendors gives Worm.Autoit label instead of one used in our dataset. Interested readers can find an example for such as case in [105].

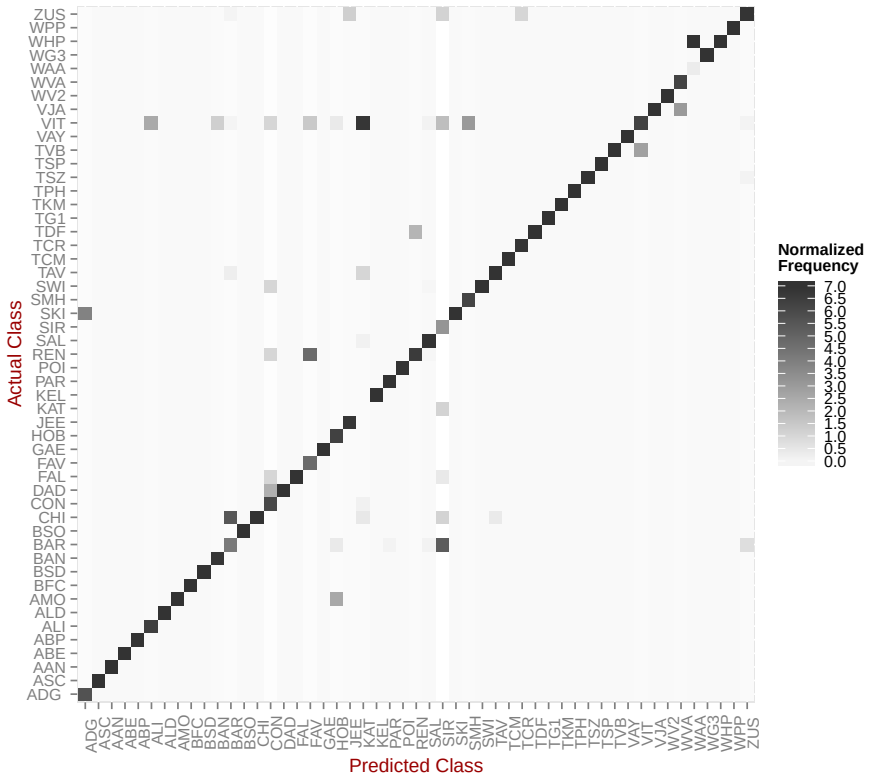


Figure 6.3: Normalized confusion matrix

Overall, the testing accuracy reaches at 92%. We analyzed the run-time behavior Virtob and realized that some of its samples generate almost similar artifacts with Kates, such as modifying same registry keys related to Internet Explorer settings and auto-start location and using same mutex names. One can inspect the following two samples's reports by their MD5 values af5ce0870db09f5f9cd9ab2bd62f42ef and 91cad3f61b7898a0a9969c1ad46883a4 from [10]. These samples belong to Kates and Virtob family, respectively.

Table 6.9 compares the proposed method and other state-of-the-art methods for classifying malware. The table shows the classification accuracy and also indicates used machine learning algorithms and features. According to Table 6.9, as the number of families increases, the classification or detection accuracy decreases. For instance, the accuracy is higher when the set of 10 families or 3 families is experimented, see for instance [200] and [201] using 3 and 10 families and reaching to 80% and 97% accuracy, respectively. However, the study using the set of 32 families present an achievement of 66.8% in accuracy ([202]). Consequently, the proposed method using a large set of families, samples and features gives more more accurate and realistic results.

6.4 Conclusion

This chapter addresses the challenge of classifying malware samples by using runtime artifacts while being robust to obfuscation. The proposed method employs runtime behaviors of an executable to build feature vector. We evaluated five different algorithms with around 18,000 current samples belonging to 48 families. CW algorithm gives the most accurate results when compared to others. Its training and testing accuracy is 94% and 92%, respectively.

In summary, this chapter has made the following contributions in the area of malware research:

- The presented dynamic malware classification approach is usable on large scale in real world.
- The results of this research indicates that runtime behavior modeling is a useful method in classifying malware.
- When compared to the recent researches, the proposed classification method achieves the highest accuracy and scale to very large data-set.

In the following chapter, we conclude the thesis by summarizing the work done and point out the future work.

Table 6.9: Comparison of proposed malware classification method with current studies

Study, Year	*Algorithm*	*Features*	*Type*	*Dataset*	*Accuracy*
[203], 2013	Online machine learning (CW, ARROW)	Features derived from URL string	Detection	1.000.000 URLs	75%
[204], 2014	Ensemble learning with Ripper, C4.5 and IBk	N-gram feature of the disassembled code	Detection	Unknown	Unknown
[200], 2014	SVM	N-gram feature of the network artifacts	Classification	Around 3.000 samples, 3 families	80%
[202], 2013	Online machine learning (algorithm unknown)	Performance monitor, system call and system call sequences	Classification	3.454 samples, 32 families	66.8%
[152], 2013	SVM, LR	Set of OS actions	Detection	5.300 malware and 100 benign	99%
[201], 2012	Information Gain & Adaboost with base classifiers	API calls and their parameters	Classification	1.368 samples, 10 families	97%
[205], 2010	AdaboostM1 with 5 base classifiers; SVM, perceptron, etc.	Function length frequency and printable string information	Detection	1400 unpacked malware and 151 benign	98%
[154], 2007	Single-linkage hierarchical clustering using normalized compress distance	Run-time artifacts	Clustering	3.700 malware samples	91%
[206], 2012	One-class SVM	APIs, strings and basic blocks	Classification	113 malware samples	78%
Our study, 2015	Online machine learning (CW, ARROW, PA-I & II, NHERD)	Run-time artifacts, IDS signatures, important API calls	Classification	17.900 malware samples, 48 families	92%

Chapter 7

Conclusion

This chapter concludes the dissertation and outlines the possible future work.

Malware has become more apparent with the exponential increase in the number of incidence and cyber attack in which individuals, large organizations and even states are involved. Currently, malware sample is equipped with advanced techniques; such as obfuscation, encrypted communication channels, sandbox evasion, etc., to fulfill their goals. Therefore, it has become almost impossible for today's security solutions to cope with the current malware samples. In the light of this remark, the primary goal of this dissertation is to classify malware samples according to their behavioral artifacts while providing scalability and automation for large scale malware analysis.

In this dissertation we have combined techniques from the discipline of malware analysis and online machine learning to build up the proposed malware classification framework. It involves the following steps:

- Large scale dynamic analysis with VirMon and Cuckoo sandboxes
- Preprocessing & feature extraction
- Modeling malware based on behavioral artifacts
- Labeling malware samples with anti virus tools
- Train sample with online machine learning algorithm
- Classify samples according to training model

- Evaluate the classification accuracy

An important contribution of this dissertation is the description of executable behavior with runtime artifacts. To identify what features are the most adequate for malware representation, we survey the known methods and schema for defining malware infection, examining their common features (indicators). Furthermore, some useful indicators related to network information is added to feature set. From this information we get a good indication of the executable behavior. For example, activities related to registry, file, process, network, IDS alert, API calls, etc. were used to build a high dimensional feature set. To address the challenge of efficient feature extraction, we encode API calls used by malware during dynamic analysis into two length long codes. By ignoring the successive API calls, the encoding schema effectively captures the semantics of the API calls while being resilient to various obfuscation techniques. Following that, an n-gram extraction is applied to the API call sequence for constructing a feature vector.

Unfortunately, our proposed method cannot resolve the problem of malware detection directly, but it provides an important step towards providing practical solutions to anti-virus companies or malware research institutes with large scale malware classification schema. The underlying assumption is that malware typically shares significant similarity in terms of tasks performed on the OS since it is potentially derived from the same code basis.

Selecting the most appropriate classifier for dataset and extracted features is the key factor to determine the system's accuracy. Comparing a set of classifier gives malware researchers to identify the classifier which satisfy their specific needs and requirements in terms of run-time efficiency and classifier accuracy. As a consequence, the proposed approach employs a set of classification algorithm to evaluate their performance while carrying out large scale experiment with behavior-based features. After testing PA-1, PA-2, AROW,NHERD and CW online classifiers with various parameters, CW shows better classification accuracy with $N = 6$ and $C = 4.0$.

7.1 Future Work

The proposed approach can be further extended in various directions, some of which are outlined below.

- The main limitation of the proposed approach is that it can only classify executable file due to the VirMon's inability to handle other file format. Considering current advanced malware is generally delivered into the target system by

exploiting client-side applications through malformed files such as PDF, Word, Excel, HTML, etc., the proposed system should support different file format as well VirMon framework. To this end, we plan to extend VirMon capabilities in order to analyze popular file formats with client side applications like Cuckoo sandbox.

- Although the VirMon and Cuckoo sandboxes provide adequate behavioral information, further research may be fruitful in three possible areas:
  - Integrate alternative dynamic analysis frameworks to investigate a sample and to compare the analysis reports for possible sandbox evasion techniques.
  - Look for additional features either dynamic or static that can be used to improve the classification accuracy.
  - Identify distinguishing malicious actions in order to efficiently reduce the feature space and improve run-time performance of the classification algorithms, especially when dealing with a large collection of malware samples.
- The rapid development of smartphone and its widespread user acceptance leads the number of malicious software targeting such platform to increase. Another future work can be done to classify malware targeting mobile platforms. To this end, researchers could use existing dynamic analysis systems for mobile applications such as [207, 208, 209, 210] or developed new one.

To summarize, our proposed methods succeeded in automatically classifying malware samples with a high degree of accuracy. We believe that our proposed system is practical and very useful in the fight against the vast amount of malware samples continually emerging everyday.

Publications

- Abdurrahman Pektaş and Tankut Acarman. A dynamic malware analyzer against virtual machine aware malicious software. In Security and Communication Networks, vol. 7, pp. 2245-2257, 2014.
- Hüseyin Tirli, Abdurrahman Pektas, Yliès Falcone and Nadia Erdogan (2013). Virmon: A Virtualization-Based Automated Dynamic Malware Analysis System, International Information Security & Cryptology Conference, available at http://www.iscturkey.org/iscold/ISCTURKEY2013/files/paper39.pdf
- Abdurrahman Pektaş, Tankut Acarman, Yliès Falcone and Jean-Claude Fernandez. Runtime-Behavior Based Malware Classification Using Online Machine Learning. In World Congress on Internet Security (WorldCIS-2015), 2015. (Accepted paper)
- Abdurrahman Pektaş, Tankut Acarman, Yliès Falcone and Jean-Claude Fernandez. Runtime-Behavior Based Malware Classification Using Online Machine Learning. In 11th EAI International Conference on Security and Privacy in Communication Networks(SecureComm-2015), 2015. (Poster paper)

Appendix

A.1 Detect VMware Version with VMware Backdoor I/O Port

```
#define MAGIC 0x564d5868  // VMware backdoor magic value = "VMXh"
#define PORT  0x5658      // VMware backdoor I/O port = "VX"
#define GETVERSION 0x0a   // Get VMware version command id = 10

#include <stdio.h>
#include <windows.h>
#include <excpt.h>

int main(int argc, char* argv[]){
unsigned int test_vmware, vmware_version;

__try {
  __asm{
    mov eax, MAGIC;
    mov ecx, GETVERSION;
    mov dx, PORT;
    in eax, dx;
    mov test_vmware, ebx
    mov vmware_version, ecx

  }
}
__except(EXCEPTION_EXECUTE_HANDLER){
  printf("An exception is occurred!!!\n");
}
```

```
26
27 if (test_vmware == 'VMXh') {
28   printf ("VMware Detected!!!\n");
29   switch(vmware_version){
30     case 1:
31       printf ("Express\n");
32       break;
33     case 2:
34       printf ("ESX\n");
35       break;
36     case 3:
37       printf ("GSX\n");
38       break;
39     case 4:
40       printf ("Workstation\n");
41       break;
42     default:
43       printf ("Unknown Version\n");
44 }
45 else
46   printf ("VMware not Detected...\n");
47 }
48
49 return 0;
50 }
```

Listing A.1: Snap Code of Red Pill Technique

A.2 Step by Step Advanced Cuckoo Installation

- Enable virtualization Technology from Bios in order to run x64 version of:
  For our Cuckoo server: Press F9 while booting and follow the following steps;
  Advanced Options -> Processor Options -> Intel Virtualization Technology ->
  Enable

- Download and install Virtualbox as malware analysis platform:

```
1 $ cd /tmp
2 $ wget http://download.virtualbox.org/virtualbox/4.3.12/virtualbox
  -4.3_4.3.12-93733~Ubuntu~raring_amd64.deb
3 $ dpkg -i virtualbox-4.3_4.3.12-93733~Ubuntu~raring_amd64.deb
4 # if there is missing and dependent libraries use the following
  command
5 # $ apt-get -f install
```

- Download and install Virtualbox Extension Pack to improve capabilities of the Virtualbox (especially VirtualBox Remote Desktop Protocol to support Remote Desktop Session)

```
$ cd /tmp
$ wget http://download.virtualbox.org/virtualbox/4.3.12/
  Oracle_VM_VirtualBox_Extension_Pack-4.3.12-93733.vbox-extpack
$ vboxmanage extpack install Oracle_VM_VirtualBox_Extension_Pack
  -4.3.12-93733.vbox-extpack
```

- Create user for vbox and modify VBOXWEB_USER defined in /etc/default/virtualbox configuration file.

```
$ adduser vbox
# edit vbox user as VBOXWEB_USER=vbox
$ vim /etc/default/virtualbox
```

- Create system start/stop links for vboxweb-service application.

```
$ update-rc.d vboxweb-service defaults
```

- Install the required packages

```
$ apt-get install gcc make apache2-mpm-prefork apache2-utils apache2
  .2-bin  apache2 apache2-doc apache2-suexec libapache2-mod-php5
  libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-ldap
  libapr1 php5-common php5-mysql  php-pear wget
```

- Download & Configure phpvirtualbox application

```
$ cd /var/www/html/;wget http://sourceforge.net/projects/
  phpvirtualbox/files/phpvirtualbox-4.3-1.zip
$ apt-get install unzip; unzip phpvirtualbox-4.3-1.zip; mv
  phpvirtualbox-4.3-1 phpvirtualbox
$ cd phpvirtualbox; cp config.php-example config.php
$vim config.php # edit username and password variable, e.g. edit var
   $username = 'defined user that runs VirtualBox service in previous
   steps';
```

- Upload an image to run in VirtualBox

```
$ scp -r /images/XPSP3x86 root@192.168.200.48:/images
$ chown -R vbox:vbox  /images/ # modify permissions of the image
```

- Browse http://192.168.200.48/phpvirtualbox/ to test the Virtualbox installation, the default username is admin, the password is admin as well.

- VLAN installation for network segregation

```
$ sudo su -c 'echo "8021q" >> /etc/modules'
$ apt-get install vlan
$ echo 1 > /proc/sys/net/ipv4/ip_forward # enable IPv4 forwarding
```

- Add virtual VLAN interface(in this case, VLAN_ID=10). Change the network interface and other values according to your scenario.

```
$ vim /etc/network/interfaces # add the following lines
auto eth1.10
iface eth1.10 inet static
    address 10.0.0.1
    netmask 255.255.255.0
    vlan-raw-device eth1
```

- Assign network interface and IP address to an analysis machine

```
$ vboxmanage modifyvm XPSP3x86-1 --nic1 bridged --bridgeadapter1
  vlan10
$ vboxmanage guestcontrol XPSP3x86-1 execute --image "C:\Windows\
  System32\netsh.exe" --username tst --password tst --wait-stdout
  interface ip set address local static 10.10.11.2 255.255.255.0
  10.10.11.1 1
$ vboxmanage guestcontrol XPSP3x86-1 execute --image "C:\Windows\
  System32\netsh.exe" --username tst --password tst --wait-stdout
  interface ip set dns "Local Area Connection" static 192.168.52.10
   primary
```

A.3 Jubatus Setup for Distributed Mode

- Install jubatus for each machine (installation procedure is given for Ubuntu Server 14.04 LTS(64-bit) as follows):
  - Write the following line to /etc/apt/sources.list.d/jubatus.list to register Jubatus apt repository to the system. deb http://download.jubat.us/apt binary/
  - Now the repo is ready to install jubatus package. Currently jubatus package is not GPG-signed. Thus, the user needs to accept the warning by typing yes to the prompt when asked.

```
$ sudo apt-get update
$ sudo apt-get install jubatus
```

  - Write jubatus binaries to the user profile to automatically load the jubatus binaries when the user logs into the system.

- Setup Zookeeper to manage jubatus servers in cluster environment
  - Configure Zookeper config file (for default intallation it is /opt/zookeeper/zookeeper-3.4.6/conf/zoo.cfg)for three jubatus servers given by their IP addresses.

```
tickTime=2000
initLimit=10
syncLimit=5
dataDir=/var/zookeeper
clientPort=2181
server.1=192.168.200.45:2888:3888
server.2=192.168.200.46:2888:3888
server.3=192.168.200.47:2888:3888
```

  - Run Zookeper service as follows:

```
$ /opt/zookeeper/zookeeper-3.4.6/bin/zkServer.sh start
JMX enabled by default
Using config: /path/to/zookeeper/bin/../conf/zoo.cfg
Starting zookeeper ...
STARTED
```

  - Register configuration file to ZooKeeper

```
jubatus1 $ jubaconfig --cmd write --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181 --
  file arow.json --name deneme --type classifier
```

  - Jubatus Proxy: proxy RPC requests from clients to servers.

```
jubatus4$  jubaclassifier_proxy --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181
```

  - Join jubatus servers to cluster

```
jubatus1$ jubaclassifier --name deneme --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181
jubatus2$ jubaclassifier --name deneme --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181
jubatus3$ jubaclassifier --name deneme --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181
```

```
jubatus5$ jubavisor --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181 --
  daemon
```

```
jubatus5$ jubactl -c start --server=jubaclassifier --type=
  classifier --name=deneme  --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181
jubatus5$ jubactl -c status --server=jubaclassifier --type=
  classifier --name=deneme  --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181
```

– Check installation

```
root@jubatus5$ jubactl -c status --server=jubaclassifier --type=
  classifier --name=deneme  --zookeeper
  =192.168.200.45:2181,192.168.200.46:2181,192.168.200.47:2181
active jubaproxy members:
192.168.200.48_9199
active jubavisor members:
192.168.200.49_9198
active deneme members:
192.168.200.45_9199
192.168.200.46_9199
192.168.200.47_9199
```

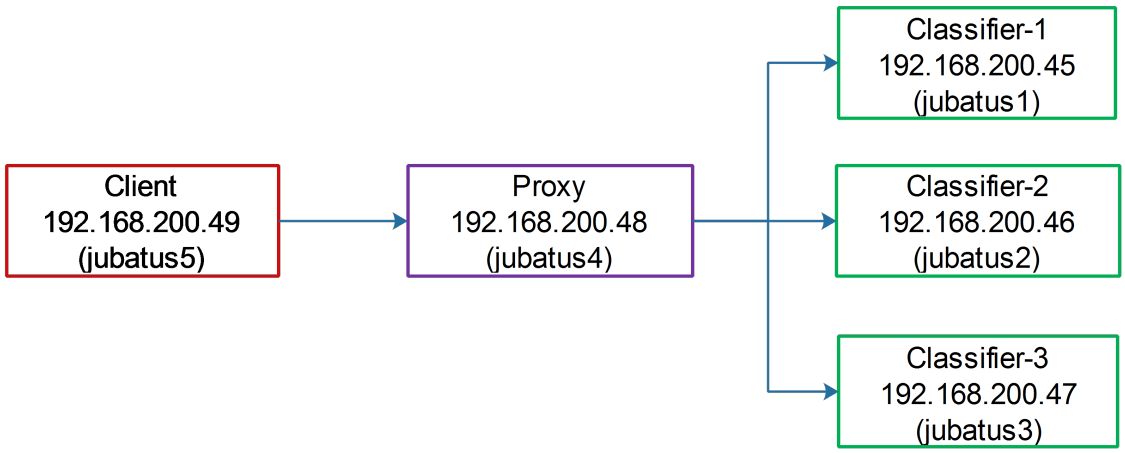


Figure A.1: Distributed Mode Jubatus

A.4 Summary of the Malicious Activities Observed in the Evaluation Set

Table A.1: Malicious activities observed in the evaluation set

Activitiy	# sample
The binary likely contains encrypted or compressed data.	98795
Installs itself for autorun at Windows startup	36976
Steals private information from local Internet browsers	22563
Collects information to fingerprint the system (MachineGuid DigitalProductId SystemBiosDate)	22347
Performs some HTTP requests	19442
Executed a process and injected code into it probably while unpacking	10010
Creates an Alternate Data Stream (ADS)	7476
The executable is compressed using UPX	3085
Connects to an IRC server possibly part of a botnet	3037
Generates some ICMP traffic	994
Detects VirtualBox through the presence of a file	662
Queries information on disks possibly for anti-virtualization	463
Creates a slightly modified copy of itself	200
Creates a windows hook that monitors keyboard input (keylogger)	184
Retrieves Windows ProductID	164
Checks the version of Bios possibly for anti-virtualization	134
Harvests credentials from local FTP client softwares	109
Checks for the presence of known devices from debuggers and forensic tools	89
Detects VirtualBox through the presence of a registry key	83
Creates known Fynloski/DarkComet mutexes	72
Checks for the presence of known windows from debuggers and forensic tools	65
Zeus P2P (Banking Trojan)	46
Operates on local firewall's policies and settings	44
Disables Windows' Registry Editor	35
Creates an autorun.inf file	19
Creates known SpyNet mutexes and/or registry changes.	16
Installs an hook procedure to monitor for mouse events	14
Installs OpenCL library probably to mine Bitcoins	10
Contacts C&C server HTTP check-in (Banking Trojan)	7
Creates known PcClient mutex and/or file changes.	3
Installs WinPCAP	2
At least one process apparently crashed during execution	2
Looks up the external IP address	2

Bibliography

[1] Maksym Schipka. "Dollars for downloading". In: *Network Security* 1 (2009), pp. 7–11.

[2] Thomas Rid and Peter McBurney. "Cyber-Weapons". In: *The RUSI Journal* 157.1 (2012), pp. 6–13. DOI: 10.1080/03071847.2012.664354. eprint: http://dx.doi.org/10.1080/03071847.2012.664354. URL: http://dx.doi.org/10.1080/03071847.2012.664354.

[3] 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.

[4] 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.

[5] Philip O'Kane, Sakir Sezer, and Kieran McLaughlin. "Obfuscation: The hidden malware". In: *IEEE Security & Privacy* 9.5 (2011), pp. 41–47.

[6] Ilsun You and Kangbin Yim. "Malware Obfuscation Techniques: A Brief Survey." In: *BWCCA*. 2010, pp. 297–300.

[7] Ashu Sharma and SK Sahay. "Evolution and Detection of Polymorphic and Metamorphic Malwares: A Survey". In: (2014).

[8] *Cisco 2014 Annual Security Report*. URL: http://www.cisco.com/web/offer/gist_ty2_asset/Cisco_2014_ASR.pdf.

[9] *AV-TEST: The independent IT-Security Institute*. URL: http://www.av-test.org/en/.

[10] Abdurrahman Pektas. *Malware Classification Based on Run-Time Behavior Using Machine Learning*. URL: http://research.pektas.in/.

[11] *Post-Exploitation with "Incognito"*. URL: http://hardsec.net/post-exploitation-with-incognito/?lang=en.

[12] Microsoft Security TechCenter. *Microsoft Security Bulletin MS08-067 - Critical*. 2008. URL: https://technet.microsoft.com/library/security/ms08-067.

[13] FireEye. *POISON IVY: Assessing Damage and Extracting Intelligence*. 2014. URL: https://www.fireeye.com/content/dam/legacy/resources/pdfs/fireeye-poison-ivy-report.pdf.

[14] FireEye. *DIGITAL BREAD CRUMBS: Seven Clues To Identifying Who's Behind Advanced Cyber Attacks*. 2014. URL: https://www.fireeye.com/content/dam/legacy/resources/pdfs/digital-bread-crumbs.pdf.

[15] Eric Chien and Gavin O'Gorman. *The Nitro Attacks: Stealing Secrets from the Chemical Industry*. 2011. URL: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the_nitro_attacks.pdf.

[16] C99Shell. *c99.php backdoor - php shell*. 2014. URL: http://corz.org/corz/c99.php.

[17] Davide Canali, Davide Balzarotti, and Aurélien Francillon. "The Role of Web Hosting Providers in Detecting Compromised Websites". In: *Proceedings of the 22Nd International Conference on World Wide Web*. WWW '13. Rio de Janeiro, Brazil: International World Wide Web Conferences Steering Committee, 2013, pp. 177–188. ISBN: 978-1-4503-2035-1. URL: http://dl.acm.org/citation.cfm?id=2488388.2488405.

[18] Natasa Suteva, Aleksandra Mileva, and Mario Loleski. "Computer forensic analisys of some web attacks". In: *World Congress on Internet Security (WorldCIS)*. 2014, pp. 42–47. DOI: 10.1109/WorldCIS.2014.7028164.

[19] *Lenovo Statement on Superfish*. 2015. URL: https://s3.amazonaws.com/isby/lenovopartnernetwork.com/upload/4/docs/lenovo-bp-statement-on-superfish.pdf.

[20] Chris Duckett. *Researchers: Lenovo laptops ship with adware that hijacks HTTPS connections*. 2015. URL: http://www.zdnet.com/article/lenovo-accused-of-pushing-superfish-self-signed-mitm-proxy/.

[21] Robert Graham. *Extracting the SuperFish certificate*. 2015. URL: http://blog.erratasec.com/2015/02/extracting-superfish-certificate.html#.VOmt2_mUfVa.

[22] Cert Polska. *ZeuS-P2P Monitoring and Analysis*. 2013. URL: http://www.cert.pl/PDF/2013-06-p2p-rap_en.pdf.

[23] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. "On the analysis of the Zeus botnet crimeware toolkit". In: *Eighth Annual International Conference on Privacy Security and Trust (PST)*. 2010, pp. 31–38. DOI: 10.1109/PST.2010.5593240.

[24] Laboratory of Cryptography and System Security (CrySyS Lab). *sKyWIper (a.k.a. Flame a.k.a. Flamer): A complex malware for targeted attacks*. 2012. URL: http://www.crysys.hu/skywiper/skywiper.pdf.

[25] Antiy Labs. *Analysis Report on Flame Worm*. 2012. URL: http://www.antiy.net/media/reports/flame-analysis.pdf.

[26] Mark Russinovich. *Sony Rootkits and Digital Rights Management Gone Too Far*. 2005. URL: http://blogs.technet.com/b/markrussinovich/archive/2005/10/31/sony-rootkits-and-digital-rights-management-gone-too-far.aspx.

[27] CERT Australia. *New Ransomware campaign*. 2013. URL: https://www.cert.gov.au/system/files/625/690/CERT_Australia_Publication_2013-72_WHITE.pdf.

[28] McAfee Labs Threat Advisory. *Ransom Cryptolocker*. 2014. URL: https://kc.mcafee.com/resources/sites/MCAFEE/content/live/PRODUCT_DOCUMENTATION/24000/PD24786/en_US/McAfee_Labs_Threat_Advisory_Ransom_Cryptolocker.pdf.

[29] Nikola Milosevic. "History of malware". In: (2013).

[30] Harold Joseph Highland. "A history of computer viruses - Introduction". In: *Computers & Security* 16.5 (1997), pp. 412 –415. ISSN: 0167-4048. DOI: http://dx.doi.org/10.1016/S0167-4048(97)82245-6. URL: http://www.sciencedirect.com/science/article/pii/S0167404897822456.

[31] John Von Neumann, Arthur W Burks, et al. "Theory of self-reproducing automata". In: *IEEE Transactions on Neural Networks* 5.1 (1966), pp. 3–14.

[32] Fred Cohen. "Computer Viruses - Theory and Experiments". In: 1984.

[33] Fred Cohen. "Computer viruses: theory and experiments". In: *Computers & security* 6.1 (1987), pp. 22–35.

[34] Aikaterinaki Niki. "Drive-by download attacks: Effects and detection methods". In: *3rd IT student conference for the next generation, University of East London*. 2009.

[35] Van Lam Le, Ian Welch, Xiaoying Gao, and Peter Komisarczuk. "Anatomy of drive-by download attack". In: *Proceedings of the Eleventh Australasian Information Security Conference*. Australian Computer Society, Inc. 2013, pp. 49–58.

[36] Julia Narvaez, Barbara Endicott-Popovsky, Christian Seifert, Chiraag Aval, and Deborah A Frincke. "Drive-by-downloads". In: *43rd Hawaii International Conference on System Sciences (HICSS)*. IEEE. 2010, pp. 1–10.

[37] Cristian Florian. *Most vulnerable operating systems and applications in 2014*. 2015. URL: http://www.gfi.com/blog/most-vulnerable-operating-systems-and-applications-in-2014/.

[38] *HB GARY Threat Report: Operation Aurora*. 2010. URL: http://hbgary.com/sites/default/files/publications/WhitePaper%20HBGary%20Threat%20Report,%20Operation%20Aurora.pdf.

[39] McAfee Labs and McAfee Foundstone Professional Services. *Protecting Your Critical Assets Lessons Learned from "Operation Aurora"*. 2010. URL: http://www.wired.com/images_blogs/threatlevel/2010/03/operationaurora_wp_0310_fnl.pdf.

[40] Lillian Ablon, Martin C Libicki, and Andrea A Golay. *Markets for Cybercrime Tools and Stolen Data: Hackers' Bazaar*. Rand Corporation, 2014.

[41] Mitre Cop. *CVE-2014-0160:Heart Bleed Vulnerability*. 2014. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160.

[42] *Heart Bleed Vulnerability*. 2014. URL: http://heartbleed.com/.

[43] Mitnick Kevin D. and Simon William L. *The Art of Intrusion: The Real Stories Behind the Exploits of Hackers, Intruders and Deceivers*. Wiley Publishing Inc., 2005.

[44] Wozniak Steve, Mitnick Kevin D., and Simon William L. *The Art of Deception: Controlling the Human Element of Security*. Rober Ipsen, 2003.

[45] Wozniak Steve, Mitnick Kevin D., and Simon William L. *Ghost in the Wires: My Adventures as the World's Most Wanted Hacker*. Back Bay Books, 2012.

[46] Common Vulnerabilities and Exposures. *CVE-2010-2568*. 2010. URL: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-2568.

[47] Rapid 7. *Apache Tomcat Manager Application Deployer Authenticated Code Execution*. 2014. URL: http://www.rapid7.com/db/modules/exploit/multi/http/tomcat_mgr_deploy.

[48] CVE Details. *Vulnerability Details: CVE-2009-3843*. 2011. URL: http://www.cvedetails.com/cve/2009-3843.

[49] Fraser Howard and Onur Komili. "Poisoned search results: How hackers have automated search engine poisoning attacks to distribute malware". In: *Sophos Technical Papers* (2010).

[50] David Y Wang, Stefan Savage, and Geoffrey M Voelker. "Juice: A Longitudinal Study of an SEO Botnet." In: *NDSS*. 2013.

[51] John P John, Fang Yu, Yinglian Xie, Arvind Krishnamurthy, and Martín Abadi. "deSEO: Combating Search-Result Poisoning." In: *USENIX Security Symposium*. 2011.

[52] Marc Fossi, Gerry Egan, Eric Johnson, Trevor Mack, Téo Adams, Joseph Blackbird, Brent Graveland, and David McKinney. "Symantec Report on Attack Kits and Malicious Websites". In: *Haettu* 5 (2011), p. 2012.

[53] Nir Kshetri. *The global cybercrime industry: economic, institutional and strategic perspectives*. Springer Science & Business Media, 2010.

[54] Fraser Howard. "Exploring the Blackhole exploit kit". In: *Sophos Technical Paper* (2012).

[55] John Oliver, S Cheng, L Manly, J Zhu, R DELA PAZ, S Sioting, and J Leopando. "Blackhole Exploit Kit: A Spam Campaign, Not a Series of Individual Spam Runs". In: *Trend Micro Incorporated Research Paper* (2012).

[56] L Gundert and M van den Berg. "A Criminal Perspective On Exploit Packs". In: *Team Cymru Business Intelligence Team* (2011).

[57] Aditya K Sood, Richard J Enbody, and Rohit Bansal. *MALWARE ANALYSIS*. 2013.

[58] Mark Tang. *Styx-like Cool Exploit Kit: How It Works*. 2013. URL: http://blog.trendmicro.com/trendlabs-security-intelligence/styx-exploit-pack-how-it-works/.

[59] Mark Tang. *Cool Exploit Kit - A new Browser Exploit Pack on the Battlefield with a "Duqu" like font drop*. 2013. URL: http://malware.dontneedcoffee.com/2012/10/newcoolek.html.

[60] *Solutionary: SERT Exploit Kit Report - A current inventory of the most popular exploit kits, the common payloads deployed and the targeted vulnerabilities*. URL: http://www.solutionary.com/_assets/pdf/sert-exploit-kit-overview-1174sr.pdf.

[61] *2014 Internet Security Threat Report*. URL: http://www.symantec.com/content/en/us/enterprise/other_resources/b-istr_main_report_v19_21291018.en-us.pdf.

[62] *Underground Hacker Markets*. 2014. URL: http://www.secureworks.com/assets/pdf-store/white-papers/wp-underground-hacking-report.pdf.

[63] *Metasploit Framework Source Code*. URL: https://github.com/rapid7/metasploit-framework.

[64] David Maynor. *Metasploit toolkit for penetration testing, exploit development, and vulnerability research*. Elsevier, 2011.

[65] Jim O'Gorman, Devon Kearns, and Mati Aharoni. *Metasploit: the penetration tester's guide*. No Starch Press, 2011.

[66] *Source code of Socail Engineering Toolkit (SET)*. URL: https://github.com/trustedsec/social-engineer-toolkit/.

[67] Nikola Pavkovic and Luka Perkov. "Social Engineering Toolkit - A systematic approach to social engineering". In: *MIPRO, Proceedings of the 34th International Convention*. IEEE. 2011, pp. 1485–1489.

[68] P Vinod, R Jaipur, V Laxmi, and M Gaur. "Survey on malware detection methods". In: *Proceedings of the 3rd Hackers' Workshop on Computer and Internet Security (IITKHACK'09)*. 2009, pp. 74–79.

[69] Jean-Marie Borello and Ludovic Mé. "Code obfuscation techniques for metamorphic viruses". In: *Journal in Computer Virology* 4.3 (2008), pp. 211–220.

[70] *Windows WMI Classes*. URL: http://msdn.microsoft.com/en-us/library/windows/desktop/aa394572(v=vs.85).aspx.

[71] *ScooyNG: The VMware detection tool*. URL: http://www.trapkit.de/research/vmm/scoopyng/index.html.

[72] *On the Cutting Edge:Thwarting Virtual Machine Detection*. URL: http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf.

[73] *Detecting the Presence of Virtual Machines Using the Local Data Table*. URL: http://tuts4you.com/request.php?2141.

[74] *Snap Code of RedPill Technique*. URL: http://charette.no-ip.com:81/programming/2009-12-30_Virtualization.

[75] Alfredo Andres Omella. *Methods for Virtual Machine Detection*. 2006. URL: http://charette.no-ip.com:81/programming/2009-12-30_Virtualization/www.s21sec.com_vmware-eng.pdf.

[76] *VMware Backdoor I/O Port*. URL: https://sites.google.com/site/chitchatvmback/backdoor.

[77] Mark Vincent Yason. *The art of unpacking*. 2008. URL: https://www.blackhat.com/presentations/bh-usa-07/Yason/Whitepaper/bh-usa-07-yason-WP.pdf.

[78] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch, 2012.

[79] JaeKeun Lee, BooJoong Kang, and Eul Gyu Im. "Evading Anti-debugging Techniques with Binary Substitution". In: *International Journal of Security & Its Applications* 8.1 (2014).

[80] Peter Ferrie. *The Ultimate Anti-Reversing Reference*. 2011. URL: http://anti-reversing.com/Downloads/Anti-Reversing/The_Ultimate_Anti-Reversing_Reference.pdf.

[81] Maik Morgenstern and Hendrik Pilz. *Useful and useless statistics about viruses and anti-virus programs*. 2010. URL: http://www.av-test.org/fileadmin/pdf/publications/caro_2010_avtest_presentation_useful_and_useless_statistics_about_viruses_and_anti-virus_programs.pdf.

[82] Fanglu Guo, Peter Ferrie, and Tzi-Cker Chiueh. "A study of the packer problem and its solutions". In: *Recent Advances in Intrusion Detection*. Springer. 2008, pp. 98–115.

[83] Seungwon Han, Keungi Lee, and Sangjin Lee. "Packed PE file detection for malware forensics". In: *2nd International Conference on Computer Science and its Applications*. IEEE. 2009, pp. 1–7.

[84] Lorenzo Martignoni, Mihai Christodorescu, and Somesh Jha. "Omniunpack: Fast, generic, and safe unpacking of malware". In: *Computer Security Applications Conference, ACSAC Twenty-Third Annual*. IEEE. 2007, pp. 431–441.

[85] Wei Yan, Zheng Zhang, and Nirwan Ansari. "Revealing packed malware". In: *IEEE Security & Privacy* 6.5 (2008), pp. 65–69.

[86] Lutz Böhne. "Pandora's bochs: Automatic unpacking of malware". PhD thesis. University of Mannheim, 2008.

[87] Colin Burgess, Fatih Kurugollu, Sakir Sezer, and Keiran McLaughlin. "Detecting packed executables using steganalysis". In: *5th European Workshop on Visual Information Processing (EUVIP)*. IEEE. 2014, pp. 1–5.

[88] Paul Royal, Mitch Halpin, David Dagon, Robert Edmonds, and Wenke Lee. "Polyunpack: Automating the hidden-code extraction of unpack-executing malware". In: *Computer Security Applications Conference ACSAC'06*. IEEE. 2006, pp. 289–300.

[89] Kevin Coogan, Saumya Debray, Tasneem Kaochar, and Gregg Townsend. "Automatic static unpacking of malware binaries". In: *16th Working Conference on Reverse Engineering*. IEEE. 2009, pp. 167–176.

[90] Guhyeon Jeong, Euijin Choo, Joosuk Lee, Munkhbayar Bat-Erdene, and Heejo Lee. "Generic unpacking using entropy analysis." In: *MALWARE*. 2010, pp. 98–105.

[91] Andreas Moser, Christopher Kruegel, and Engin Kirda. "Limits of static analysis for malware detection". In: *Twenty-third annual Computer security applications conference*. IEEE. 2007, pp. 421–430.

[92] Michael Ligh, Steven Adair, Blake Hartstein, and Matthew Richard. *Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code*. Wiley Publishing, 2010.

[93] *Anubis - Malware Analysis for Unknown Binaries*. URL: https://anubis.iseclab.org/.

[94] Carsten Willems, Thorsten Holz, and Felix Freiling. "Toward automated dynamic malware analysis using cwsandbox". In: *IEEE Security & Privacy* 2 (2007), pp. 32–39.

[95] Norman ASA. *Norman sandbox whitepaper*. Tech. rep. Technical report, 2003.

[96] *RegShot*. URL: http://sourceforge.net/projects/regshot/.

[97] *Automated Malware Analysis - Cuckoo Sandbox*. URL: http://cuckoosandbox.org/.

[98] Dawn Song et al. "BitBlaze: A new approach to computer security via binary analysis". In: *Information systems security*. Springer, 2008, pp. 1–25.

[99] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. *TTAnalyze: A tool for analyzing malware*. na, 2006.

[100] Fabrice Bellard. "QEMU, a Fast and Portable Dynamic Translator." In: *USENIX Annual Technical Conference, FREENIX Track*. 2005, pp. 41–46.

[101] Min Gyung Kang, Pongsin Poosankam, and Heng Yin. "Renovo: A hidden code extractor for packed executables". In: *Proceedings of the ACM workshop on Recurring malcode*. ACM. 2007, pp. 46–53.

[102] Zhenkai Liang, Heng Yin, and Dawn Song. "HookFinder: Identifying and understanding malware hooking behaviors". In: *Department of Electrical and Computing Engineering* (2008), p. 41.

[103] *Attacks on Virtual Machine Emulators*. 2007. URL: https://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf.

[104] *Joanna Rutkowska's Blog*. URL: http://theinvisiblethings.blogspot.com.

[105] *VirusTotal: a free service that analyzes suspicious files and URLs*. URL: https://www.virustotal.com/.

[106] *PEiD: common packers detector*. URL: http://www.aldeid.com/wiki/PEiD.

[107] *Pefile: a Python module to read and work with PE (Portable Executable) files*. URL: https://code.google.com/p/pefile/.

[108] *Pestudio: a tool that performs the static analysis of 32-bit and 64-bit Windows executable files*. URL: http://www.winitor.com/.

[109] *IDApro: multi-processor disassembler and debugger*. URL: https://www.hex-rays.com/products/ida/.

[110] *Dependency Walker*. URL: http://www.dependencywalker.com/.

[111] *PEView*. URL: http://www.aldeid.com/wiki/PEView.

[112] *PEBrowse64 Professional*. URL: http://www.smidgeonsoft.prohosting.com/pebrowse-pro-file-viewer.html.

[113] *PE Explorer*. URL: http://www.heaventools.com/overview.htm.

[114] *Resource Hacker*. URL: http://www.angusj.com/resourcehacker/.

[115] Mark Russinovich and Bryce Cogswell. *Process Monitor v3.1*. 2014. URL: https://technet.microsoft.com/en-us/sysinternals/bb896645.

[116] Mark Russinovich. *Process Explorer v16.04*. 2014. URL: https://technet.microsoft.com/en-us/sysinternals/bb896653.

[117] Mark Russinovich and Bryce Cogswell. *Autoruns for Windows v13.01*. 2015. URL: https://technet.microsoft.com/en-us/sysinternals/bb963902.

[118] *Process Hacker*. URL: http://processhacker.sourceforge.net/.

[119] *Capture-BAT*. URL: https://www.honeynet.org/node/315.

[120] Christian Seifert, Ramon Steenson, Ian Welch, Peter Komisarczuk, and Barbara Endicott-Popovsky. "Capture - A behavioral analysis tool for applications and documents". In: *digital investigation* 4 (2007), pp. 23–30.

[121] *Volatility - An advanced memory forensics framework*. URL: https://github.com/volatilityfoundation/volatility.

[122] Hüseyin Tirli, Abdurrahman Pektas, Yliès Falcone, and Nadia Erdogan. "Virmon: A Virtualization-Based Automated Dynamic Malware Analysis System". In: *International Information Security & Crptology Conference*. 2013. URL: http://www.iscturkey.org/iscold/ISCTURKEY2013/files/paper39.pdf.

[123] *Wireshark - Network Analyzer*. URL: https://www.wireshark.org/.

[124] *Microsoft Network Monitor 3.4*. URL: http://www.microsoft.com/en-us/download/details.aspx?id=4865.

[125] *OllyDbg*. URL: http://www.ollydbg.de/.

[126] *Immunity Debugger*. URL: http://debugger.immunityinc.com/.

[127] *WinDbg*. URL: http://www.windbg.org/.

[128] VirusTotal. *Anti-Virus Scan for Zeus Sample*. 2013. URL: http://bit.ly/1iSuvGo.

[129] Osman Pamuk and Necati Ersen Siseci. *A Zeus Sample Analysis (FatMal)*. 2012. URL: http://www.bilgiguvenligi.gov.tr/zararli-yazilimlar/fatura-zararli-yazilimi-fatmal.html.

[130] Gerald J Tesauro, Jeffrey O Kephart, and Gregory B Sorkin. "Neural networks for computer virus recognition". In: *IEEE expert* 11.4 (1996), pp. 5–6.

[131] William Arnold and Gerald Tesauro. "Automatically generated Win32 heuristic virus detection". In: *Proceedings of the international virus bulletin conference*. 2000.

[132] Matthew G Schultz, Eleazar Eskin, Erez Zadok, and Salvatore J Stolfo. "Data mining methods for detection of new malicious executables". In: *IEEE Symposium on Security and Privacy*. IEEE. 2001, pp. 38–49.

[133] Tony Abou-Assaleh, Nick Cercone, Vlado Keselj, and Ray Sweidan. "N-gram-based detection of new malicious code". In: *Proceedings of the 28th Annual International Computer Software and Applications Conference*. Vol. 2. IEEE. 2004, pp. 41–42.

[134] Jeremy Z Kolter and Marcus A Maloof. "Learning to detect malicious executables in the wild". In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2004, pp. 470–478.

[135] Abdurrahman Pektaş, Mehmet Eriş, and Tankut Acarman. "Proposal of n-gram based algorithm for malware classification". In: *SECURWARE, The Fifth International Conference on Emerging Security Information, Systems and Technologies*. 2011, pp. 14–18.

[136] M Zubair Shafiq, S Momina Tabish, Fauzan Mirza, and Muddassar Farooq. "Peminer: Mining structural information to detect malicious executables in real-time". In: *Recent advances in intrusion detection*. Springer. 2009, pp. 121–141.

[137] *VX Heavens Virus Collection*. 2015. URL: http://vxheaven.org/vl.php.

[138] *Malfease Project Malware Dataset*. 2015. URL: http://malfease.oarci.net.

[139] Christian Wressnegger, Guido Schwenk, Daniel Arp, and Konrad Rieck. "A close look on n-grams in intrusion detection: anomaly detection vs. classification". In: *Proceedings of the ACM workshop on Artificial intelligence and security*. ACM. 2013, pp. 67–76.

[140] Ke Wang and Salvatore J Stolfo. "Anomalous payload-based network intrusion detection". In: *Recent Advances in Intrusion Detection*. Springer. 2004, pp. 203–222.

[141] Roberto Perdisci, Davide Ariu, Prahlad Fogla, Giorgio Giacinto, and Wenke Lee. "McPAD: A multiple classifier system for accurate payload-based anomaly detection". In: *Computer Networks* 53.6 (2009), pp. 864–881.

[142] Ke Wang, Janak J Parekh, and Salvatore J Stolfo. "Anagram: A content anomaly detector resistant to mimicry attack". In: *Recent Advances in Intrusion Detection*. Springer. 2006, pp. 226–248.

[143] Konrad Rieck, Tammo Krueger, and Andreas Dewald. "Cujo: efficient detection and prevention of drive-by-download attacks". In: *Proceedings of the 26th Annual Computer Security Applications Conference*. ACM. 2010, pp. 31–39.

[144] Christopher Krügel, Thomas Toth, and Engin Kirda. "Service specific anomaly detection for network intrusion detection". In: *Proceedings of the 2002 ACM symposium on Applied computing*. ACM. 2002, pp. 201–208.

[145] Guillaume Bonfante, Matthieu Kaczmarek, and Jean-Yves Marion. "Control flow to detect malware". In: *Inter-Regional Workshop on Rigorous System Development and Analysis*. 2007.

[146] Silvio Cesare and Yang Xiang. "Malware variant detection using similarity search over sets of control flow graphs". In: *IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE. 2011, pp. 181–189.

[147] Yuxin Ding, Wei Dai, Shengli Yan, and Yumei Zhang. "Control flow-based opcode behavior analysis for Malware detection". In: *Computers & Security* 44 (2014), pp. 65–74.

[148] *Offensive Computing*. URL: http://www.offensivecomputing.net/.

[149] Younghee Park, Douglas Reeves, Vikram Mulukutla, and Balaji Sundaravel. "Fast malware classification by automated behavioral graph matching". In: *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*. ACM. 2010, p. 45.

[150] Zahra Salehi, Mahboobeh Ghiasi, and Ashkan Sami. "A miner for malware detection based on api function calls and their arguments". In: *16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP)*. IEEE. 2012, pp. 563–568.

[151] Zahra Salehi, Ashkan Sami, and Mahboobe Ghiasi. "Using feature generation from API calls for malware detection". In: *Computer Fraud & Security* 2014.9 (2014), pp. 9–18.

[152] Mahinthan Chandramohan, Hee Beng Kuan Tan, Lionel C Briand, Lwin Khin Shar, and Bindu Madhavi Padmanabhuni. "A scalable approach for malware detection through bounded feature space behavior modeling". In: *IEEE/ACM 28th International Conference on Automated Software Engineering (ASE)*. IEEE. 2013, pp. 312–322.

[153] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. "Scalable, Behavior-Based Malware Clustering." In: *NDSS*. Vol. 9. Citeseer. 2009, pp. 8–11.

[154] Michael Bailey, Jon Oberheide, Jon Andersen, Z Morley Mao, Farnam Jahanian, and Jose Nazario. "Automated classification and analysis of internet malware". In: *Recent advances in intrusion detection*. Springer. 2007, pp. 178–197.

[155] Jiyong Jang, David Brumley, and Shobha Venkataraman. "Bitshred: feature hashing malware for scalable triage and semantic analysis". In: *Proceedings of the 18th ACM conference on Computer and communications security*. ACM. 2011, pp. 309–320.

[156] Konrad Rieck, Thorsten Holz, Carsten Willems, Patrick Düssel, and Pavel Laskov. "Learning and classification of malware behavior". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2008, pp. 108–125.

[157] Aziz Mohaisen, Andrew G West, Allison Mankin, and Omar Alrawi. "Chatter: Classifying malware families using system event ordering". In: *IEEE Conference on Communications and Network Security (CNS)*. IEEE. 2014, pp. 283–291.

[158] Aziz Mohaisen and Omar Alrawi. "Amal: High-fidelity, behavior-based automated malware analysis and classification". In: *Information Security Applications*. Springer, 2014, pp. 107–121.

[159] Manuel Egele, Theodoor Scholte, Engin Kirda, and Christopher Kruegel. "A survey on automated dynamic malware-analysis techniques and tools". In: *ACM Computing Surveys (CSUR)* 44.2 (2012), p. 6.

[160] Digit Oktavianto and Iqbal Muhardianto. *Cuckoo Malware Analysis*. Packt Publishing Ltd, 2013.

[161] *The Honeynet Project*. URL: http://honeynet.org/.

[162] Abdurrahman Pektas and Tankut Acarman. "A dynamic malware analyzer against virtual machine aware malicious software". In: *Security and Communication Networks* 7.12 (2014), pp. 2245–2257. ISSN: 1939-0122. DOI: 10.1002/sec.931. URL: http://dx.doi.org/10.1002/sec.931.

[163] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. "Pin: building customized program analysis tools with dynamic instrumentation". In: *Acm Sigplan Notices*. Vol. 40. 6. ACM. 2005, pp. 190–200.

[164] Intel Corp. *Pin - A Dynamic Binary Instrumentation Tool*. 2015. URL: https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool.

[165] *Anti-virus Tracker*. URL: http://www.avtracker.info/.

[166] Microsoft Cop. *Kernel Patch Protection: Frequently Asked Questions*. URL: http://msdn.microsoft.com/en-us/library/windows/hardware/gg487353.aspx.

[167] Microsoft Cop. *Writing Preoperation and Postoperation Callback Routines*. URL: http://msdn.microsoft.com/en-us/library/windows/hardware/ff557334(v=vs.85).aspx.

[168] Microsoft Cop. *What is the registry?* URL: http://windows.microsoft.com/en-id/windows-vista/what-is-the-registry.

[169] Necati Ersen Siseci, Bakir Emre, and Hüseyin Tirli. *Case Study: Malicious Activity in the Turkish Network*. URL: http://www.syssec-project.eu/media/page-media/3/syssec-d5.3-TurkishNetworkCaseStudy.pdf.

[170] Xuxian Jiang and Xinyuan Wang. ""Out-of-the-box" Monitoring of VM-based High-Interaction Honeypots". In: *Recent Advances in Intrusion Detection*. Springer. 2007, pp. 198–218.

[171] Vincent Nicomette, Mohamed Kaâniche, Eric Alata, and Matthieu Herrb. "Set-up and deployment of a high-interaction honeypot: experiment and lessons learned". In: *Journal in Computer Virology* 7.2 (2011), pp. 143–157.

[172] *Oracle VM Virtual Box*. URL: https://www.virtualbox.org/.

[173] Lucas Nussbaum, Pierre Neyron, and Olivier Richard. "On robust covert channels inside DNS". In: *Emerging Challenges for Security, Privacy and Trust*. Springer, 2009, pp. 51–62.

[174] Cheng Qi, Xiaojun Chen, Cui Xu, Jinqiao Shi, and Peipeng Liu. "A bigram based real time DNS tunnel detection approach". In: *Procedia Computer Science* 17 (2013), pp. 852–860.

[175] Alessio Merlo, Gianluca Papaleo, Stefano Veneziano, and Maurizio Aiello. "A comparative performance evaluation of DNS tunneling tools". In: *Computational Intelligence in Security for Information Systems*. Springer, 2011, pp. 84–91.

[176] Aleksandar Lazarevic, Vipin Kumar, and Jaideep Srivastava. "Intrusion detection: A survey". In: *Managing Cyber Threats*. Springer, 2005, pp. 19–78.

[177] *Suricata IDS*. URL: http://suricata-ids.org.

[178] *The Bro Network Security Monitor*. URL: http://www.bro.org/.

[179] Bing Chen, Joohan Lee, and Annie S Wu. "Active event correlation in Bro IDS to detect multi-stage attacks". In: *Fourth IEEE International Workshop on Information Assurance*. IEEE. 2006.

[180] Pedram Amini, Reza Azmi, and MuhammadAmin Araghizadeh. "Botnet Detection using NetFlow and Clustering". In: *Advances in Computer Science: an International Journal* 3.2 (2014), pp. 139–149.

[181] Leyla Bilge, Davide Balzarotti, William Robertson, Engin Kirda, and Christopher Kruegel. "Disclosure: detecting botnet command and control servers through large-scale netflow analysis". In: *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM. 2012, pp. 129–138.

[182] *Antivirus scan results for 186c097b9d85b3501efcc4d8d374afe1 in VirusTotal*. URL: http://tinyurl.com/oe8nqzl/.

[183] Andrew J White. "Identifying the unknown in user space memory". PhD thesis. Queensland University of Technology, 2013.

[184] *Antivirus scan results for 76387075c90533aad14e82a5d94e8486 in VirusTotal*. URL: http://tinyurl.com/obr2luj/.

[185] *Sophisticated Indicators for the Modern Threat Landscape: An Introduction to OpenIOC*. URL: http://openioc.org/resources/An_Introduction_to_OpenIOC.pdf.

[186] MITRE Corporation. *MITRE Corporation. The MAEC Language Version 4.0.1 Overview*. URL: http://maec.mitre.org/about/docs/MAEC_Overview.pdf.

[187] *Mandiant Corp. - Security Consulting Services*. URL: https://www.mandiant.com/.

[188] *Mandiant OpenIOC Editor*. URL: http://www.mandiant.com/resources/download/ioc-editor/.

[189] *MITRE Corporation*. URL: http://www.mitre.org/.

[190] *Emerging Threats - ETPro Ruleset*. URL: http://devclean.emergingthreats.net/products/etpro-ruleset/.

[191] Mark Dredze, Koby Crammer, and Fernando Pereira. "Confidence-weighted linear classification". In: *Proceedings of the 25th international conference on Machine learning*. ACM. 2008, pp. 264–271.

[192] Koby Crammer and Daniel D Lee. "Learning via gaussian herding". In: *Advances in neural information processing systems*. 2010, pp. 451–459.

[193] Steven CH Hoi, Jialei Wang, and Peilin Zhao. "Libol: A library for online learning algorithms". In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 495–499.

[194] Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. "Moa: Massive online analysis". In: *The Journal of Machine Learning Research* 11 (2010), pp. 1601–1604.

[195] *Virusshare: Malware Sharing Platform*. URL: http://www.virusshare.com/.

[196] Aziz Mohaisen and Omar Alrawi. "Av-meter: An evaluation of antivirus scans and labels". In: *Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 2014, pp. 112–131.

[197] Aziz Mohaisen, Omar Alrawi, Matt Larson, and Danny McPherson. "Towards a methodical evaluation of antivirus scans and labels". In: *Information Security Applications*. Springer, 2014, pp. 231–241.

[198] Koby Crammer, Ofer Dekel, Joseph Keshet, Shai Shalev-Shwartz, and Yoram Singer. "Online passive-aggressive algorithms". In: *The Journal of Machine Learning Research* 7 (2006), pp. 551–585.

[199] Koby Crammer, Alex Kulesza, and Mark Dredze. "Adaptive regularization of weight vectors". In: *Advances in neural information processing systems*. 2009, pp. 414–422.

[200] A. Mohaisen, A.G. West, A. Mankin, and O. Alrawi. "Chatter: Classifying malware families using system event ordering". In: *IEEE Conference on Communications and Network Security (CNS)*. 2014, pp. 283–291. DOI: 10.1109/CNS.2014.6997496.

[201] Veelasha Moonsamy, Ronghua Tian, and Lynn Batten. "Feature reduction to speed up malware classification". In: *Information security technology for applications*. Springer, 2012, pp. 176–188.

[202] Raymond Canzanese, Moshe Kam, and Spiros Mancoridis. "Toward an automatic, online behavioral malware classification system". In: *IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*. IEEE. 2013, pp. 111–120.

[203] Min-Sheng Lin, Chien-Yi Chiu, Yuh-Jye Lee, and Hsing-Kuo Pao. "Malicious URL filtering - A big data application". In: *IEEE International Conference on Big Data*. IEEE. 2013, pp. 589–596.

[204] Ms Jyoti Landage and MP Wankhade. "Malware Detection with Different Voting Schemes". In: *COMPUSOFT, An international journal of advanced computer technology (IJACT)* 3.1 (2014), pp. 2320–0790.

[205] Rafiqul Islam, Ronghua Tian, Lynn Batten, and Steve Versteeg. "Classification of malware based on string and function feature selection". In: *Cybercrime and Trustworthy Computing, Workshop*. IEEE. 2010, p. 917.

[206] Yang Zhong, Hirofumi Yamaki, and Hiroki Takakura. "A malware classification method based on similarity of function structure". In: *IEEE/IPSJ 12th International Symposium on Applications and the Internet (SAINT)*. IEEE. 2012, pp. 256–261.

[207] Anthony Desnos and Patrik Lantz. *DroidBox: An Android Application Sandbox for Dynamic Analysis*. URL: https://code.google.com/p/droidbox/.

[208] Thomas Eder, Michael Rodler, Dieter Vymazal, and Markus Zeilinger. "ANANAS-A Framework For Analyzing Android Applications". In: *Eighth International Conference on Availability, Reliability and Security (ARES)*. IEEE. 2013, pp. 711–719.

[209] LK Yan and H Yin. "DroidScope: Seamlessly Reconstructing the OS and Dalvik". In: *Proceedings of USENIX Security Symposium. USENIX Association*. 2012.

[210] Lukas Weichselbaum, Matthias Neugschwandtner, Martina Lindorfer, Yanick Fratantonio, Victor van der Veen, and Christian Platzer. "Andrubis: Android Malware Under The Magnifying Glass". In: *Vienna University of Technology, Tech. Rep. TRISECLAB* (2014).