UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

Dissection of Modern Malicious Software

Fábio José Sousa Rodrigues

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Doutor Pedro R. M. Inácio

Covilhã, Outubro de 2014

Acknowledgements

It is difficult to find the right words for demonstrating my deep apreciation for the persons I am about to acknowledge.

First of all, I would like to thank my parents, Maria Fernanda and José Manuel, for supporting me financially and affectively and for helping me to accomplish this goal in my life. Their love is a blessing that will never be forgotten. Next, I would like to thank my supervisor, Professor Pedro Inácio, for accepting me for this masters programme, even though he already knew me from the final degree project. I would like to acknowledge him for always believing in me, for all the knowledge he passed to me and all the time spent with me, following the correct development of this programme and writing of this dissertation.

A kind word is also due to my friends, who helped me along this phase of my life, allowing me to surpass this challenging task.Thank you for always believing in me, for all the support and confidence: Pedro Tavares, Sérgio Costa and Emanuel Grancho.

I could not end this section without issuing the deepest thanks to the Multimedia Signal Processing Covilhã Group (MSP-CV) of Instituto de Telecomunicações, for providing me with the working conditions to successfully accomplish the objectives of this programme and to all its members, specially to Fábio Teixeira, Miguel Neto, Francisco Vigário and Telmo Reinas, for all the help and support they always gave me.

To all of the aforementioned, and to those I may have forgot to mention, but that helped me along the way,

Thank you!

Resumo

O crescimento exponencial do número de amostras de software malicioso, conhecido na gíria informática como *malware*, constitui atualmente uma das maiores preocupações dos profissionais de cibersegurança. São vários os objetivos dos criadores deste tipo de software e a forma cada vez mais sofisticada como os mesmos são alcançados. O aumento da computação e capacidade de armazenamento, bem como a globalização, têm contribuído para este crescimento, e têm alimentado toda uma indústria dedicada ao desenvolvimento, venda e melhoramento de sistemas ou soluções de segurança, recuperação, mitigação e prevenção de incidentes relacionados com *malware*. O sucesso destes sistemas depende normalmente da análise detalhada, feita muitas vezes por humanos, de peças de *malware* capturadas no seu ambiente de atuação. Esta análise compreende a procura de padrões ou de comportamentos anómalos que possam servir de assinatura para identificar ou contra-atacar essas ameaças.

Esta dissertação aborda a problemática da análise e dissecação de *malware*. O trabalho que lhe está subjacente tinha como objetivos estudar e compreender as técnicas utilizadas por este tipo de software hoje em dia, bem como as que são utilizadas por especialistas nessa análise, de forma a conduzir uma investigação detalhada e a produzir documentação estruturada sobre pelo menos uma amostra de *malware* moderna. O trabalho focou-se, sobretudo, em *malware* desenvolvido para os sistemas operativos da família Microsoft Windows para computadores de secretária. Após um breve estudo ao estado da arte, a dissertação apresenta as classificações de *malware* encontradas na literatura técnica da especialidade, principalmente usada pela indústria, resultante de um esforço de unificação das mesmas. São também apresentadas algumas das famílias de *malware* mais detetadas da atualidade, inicialmente através de uma tabela e, posteriormente, através de uma árvore geneológica, com algumas das variantes de cada uma das famílias descritas previamente. Esta árvore fornece uma perspetiva interessante sobre *malware* e constitui uma das contribuições deste programa de mestrado.

Ainda no âmbito da descrição de funcionalidades e comportamentos do *malware*, são expostas, com algum detalhe, algumas técnicas avançadas com as quais os programas maliciosos mais modernos são por vezes munidos com o intuito a facilitar a sua propagação e execução, dificultando a sua deteção. A descrição evolui para a apresentação dos conceitos adjacentes à deteção e combate ao *malware* moderno, assim como para uma pequena introdução ao tema principal deste trabalho. A análise e dissecação de duas amostras de *malware* moderno surgem nos capítulos finais da dissertação. Ao *malware* conhecido por *Stuxnet* é feita a análise básica estática, enquanto que ao *Trojan Banker Tinba/zusy* é feita e demonstrada a análise dinâmica básica e avançada. Os resultados desta parte são demonstrativos do grau de sofisticação e perigosidade destas amostras e das dificuldades associadas a estas tarefas.

Palavras-chave

Análise Estática, Análise Dinâmica, Dissecação, *Malware*, Software Malicioso, *Stuxnet*, *Tinba*/*zusy*, Trojano, Vírus Informático, *Worm*, Antivírus, Famílias de *Malware*, Comportamento de *Malware*.

Resumo alargado

Introdução

Esta secção apresenta um resumo alargado da restante dissertação. Para isso, foi dividida em dez subsecções, cujo titulo identifica o capitulo ou secção respetiva no corpo do documento, em que cada uma resume uma parte do mesmo.

Inicialmente, começa-se por realizar a exposição dos principais objetivos desta dissertação, bem como o seu enquadramento, abordagem e principais contribuições. De seguida alguns conceitos base sobre o *malware* são descritos, seguidos pela exposição de algumas famílias e informações de malware. Posteriormente, são descritas algumas dicas, sistemas e ferramentas para combate ao malware, bem como a descrição detalhada da análise básica e avançada realizada no âmbito deste programa de mestrado. Na antepenúltima secção são expostas as principais conclusões e definidas as linhas condutoras para trabalho futuro. Posteriormente surgem algumas informações complementares em anexo.

Objectivos, Enquadramento e Abordagem

O principal problema abordado foi o de analisar peças de malware modernas, tipicamente bastante complexas. Como este problema é por vezes difícil, definiu-se como principal objetivos a análise de amostras de *malware* modernas seguindo um conjunto de procedimentos descritos na literatura. Optou-se por realizar a análise de malware, desenhado especificamente para uma única família de sistemas operativos, nomeadamente o Microsoft Windows.

Os objetivos principais são o estudo e documentação de uma, ou mais, amostras de *malware* moderno. Como objetivos secundários, definiu-se a obtenção de conhecimento nas tecnologias mais recentes usadas pelos criadores de *malware* e, adicionalmente, a criação de uma árvore de famílias de *malware*. Finalmente, eram também pretendidas algumas conclusões com linhas orientadoras para a implementação de mecanismos de deteção de *malware*.

A abordagem seguida consistiu, numa primeira fase, na realização de um estudo das áreas relevantes para a realização deste trabalho. Esta tarefa compreendeu um estudo de alguns artigos e relatórios científicos e não científicos. Após esta análise, um relatório sobre o estado da arte foi produzido. A árvore de famílias de *malware* foi conseguida depois deste estudo. De igual forma, foi prevista a análise de uma ou mais amostras de malware. Antes de proceder à análise, foi montado um ambiente seguro de análise.

Contribuições Principais

As principais contribuições deste programa de mestrado são: (i) uma revisão do estado da arte, incluída no Capítulo 2, (ii) uma árvore genealógica de 23 famílias de *malware*, discutida no

Capítulo 3, (iii) a explicação de como instalar um ambiente seguro de análise de *malware* no Capítulo 5, e (iv), a análise básica e avançada de amostras de *malware*, incluída nos Capítulos 5 e 6.

Esta dissertação descreve vários passos que um analista forense realiza, aquando da análise de *malware*. A amostra do `Tinba/Zusy`, analisada no âmbito deste programa de mestrado, é recente e não há ainda relatórios da sua análise na literatura com o mesmo nível de detalhe como o que é aqui incluído.

Conceitos Base sobre *Malware* e Revisão do Estado da Arte

Este Capítulo inicia o estudo do estado da arte, começando por uma análise a artigos e relatórios relacionados com o tema deste programa de mestrado. Depois desta análise, as duas classificações de *malware* mais utilizadas pelos vários membros da indústria atualmente é descrita com detalhe, sendo um sumário incluído no final nas tabelas 2.1 and 2.2. A classificação mais antiga (baseada em comportamento), que se divide em três tipos de *malware*, é apresentada primeiro:

- **ferramenta maliciosa** , que engloba as subcategorias de *Constructor*, *Denial of Service (DoS)*, *Spoofer*, 4 diferentes tipos de *Flooders* , *Hoax*, *VirTool*, *HackTool*, *Backdoor* e *Exploit*;

- **programa malicioso** , que poderá ser dividido consoante o alvo em *malware* em massa ou *malware direcionado*, e ainda, consoante ao comportamento em 4 tipos diferentes, nomeadamente *Rootkit*, *Trojan*, *Virus*, *Worms* e *Packers*; e

- ***adware*, *pornware* ou *riskware*.**

Seguidamente é apresentada a classificação alternativa, com base em tendências do *malware*. Esta, engloba as classificações de *Crimeware*, *Spyware*, *Ransomware* e *Bot-Clients*.

Famílias e Informação sobre *Malware*

Neste Capítulo surge a explicação da convenção adotada pela maioria das companhias existentes na indústria de segurança para a nomeação de ameaças. Foi também concretizado um levantamento das 10 amostras de *malware* mais detetadas nos últimos tempos pelas empresas Microsoft, Kaspersky e ESET, aqui representado sob a forma de uma tabela, ordenada da ameaça mais detetada para a menos detetada. São, também, mencionados alguns detalhes referentes a cada uma das ameaças previamente enumeradas nas tabelas 3.1, 3.2, 3.3 e 3.4. A árvore de família foi construída para estas ameaças. Também neste Capítulo, foi realizado um estudo detalhado do conjunto de técnicas usadas para *ocultar* o *malware* em tempo de execução e dificultar a sua análise. Foram explicados alguns comportamentos técnicos do *malware*, algumas técnicas de ocultação do *malware* no sistema, técnicas de codificação do código, mecanismos que impedem a reversão do código do ficheiro executável ou das bibliotecas a ele adjacentes

e as técnicas que permitem defraudar qualquer tentativa de análise quando detetada a sua execução num ambiente virtual.

Combater o *Malware*

No Capítulo de combate ao *malware*, foram abordadas as técnicas mais comuns para combater a ocorrência de uma possível infeção a um sistema informático. Neste capítulo são apontadas algumas políticas, que, se seguidas, poderão limitar a possibilidade de ocorrência de uma infeção. Ainda neste capítulo, algumas das tecnologias de combate a, e deteção de, *malware* são apresentadas, nomeadamente, *firewalls*, *Intrusion Detection Systems (IDSs)*, *Intrusion Protection Systems (IPSs)* e programas antivírus. Ainda neste capítulo é feita uma breve introdução às técnicas e ferramentas de dissecação de *malware*.

Análise Básica Estática e Dinâmica

As técnicas usadas numa primeira abordagem à análise de malware, dizem-se básicas estáticas e dinâmicas. Nesta fase, é possível a obtenção de um conjunto de informação que pode ou não vir a ser útil numa fase posterior da análise. Este Capítulo tem como objetivo a discussão das técnicas de análise consideradas básicas. Para isso, são explicadas ferramentas e técnicas que permitem essa análise. Após a apresentação das ferramentas e técnicas associadas a este tipo de análises, é efetuada uma primeira demonstração da análise básica estática a uma amostra de *malware*, conhecida como `Stuxnet`, utilizando as ferramentas previamente descritas. De seguida, é apresentada uma proposta para um ambiente de análise seguro. Posteriormente é documentada uma análise básica dinâmica detalhada à amostra obtida do `Trojan Tinba/Zusy`, expondo os detalhes e conhecimentos obtidos desta análise numa fase final do capítulo.

Análise Avançada

Este capítulo possui uma estrutura semelhante ao capítulo anterior, mas apresenta a análise avançada. No âmbito desta análise, foi feita uma pequena explicação da abordagem tomada no inicio do Capítulo, para depois serem apresentadas as técnicas e tecnologias utilizadas no decorrer da análise avançada. Posteriormente foi efetuada uma análise avançada com base na abordagem proposta à amostra de malware do `Trojan Tinba/Zusy`, expondo os detalhes obtidos no final do capítulo.

Conclusões e Trabalho Futuro

O fator que ainda contribui consideravelmente para a infeção dos sistemas continua a ser o fator humano, de que tantas ameaças se valem para conseguir proliferar. O estado da cibersegurança atualmente demonstra que não existem sistemas infalíveis, enfatizando a necessidade de promover formações aos utilizadores de forma a diminuir ao máximo o risco de infeção. As infeções derivadas de *malware* tornaram-se mais comuns e, hoje em dia, estima-se que apareçam cerca de 315000 novas ameaças por dia [Kas13]. Quando comparado este valor com o do ano 2012, é

clara a força e o crescimento do *software* malicioso. Com o elevado número de ameaças detetadas diariamente, pode-se concluir acerca da complexidade associada à análise de todas estas ameaças por parte dos especialistas.

No que toca a mecanismos de deteção de ameaças, os antivírus baseados na *Cloud* apresentam uma alternativa moderna às soluções tradicionais. Apesar das suas vantagens ((i) menor custo computacional no computador do utilizador, (ii) base de dados atualizadas na *Cloud*, e (iii) o sistema ser gratuito), torna-se um mecanismo pouco viável para quem privilegia a confidencialidade e a privacidade dos seus dados.

O ato de nomear uma ameaça cabe a quem a descobre. Atendendo à falta de um *standard* para a nomeação de ameaças, a mesma ameaça pode ter diferentes nomes nas diversas companhias especializadas do setor.

Encontrar amostras de *malware* que pudessem ser utilizadas no âmbito deste mestrado revelou-se um desafio. O misticismo que rodeia esta área contribuiu também para este desafio. Apesar de recentes, as amostras de *malware* encontradas, demonstraram ter alguns comportamentos que dificultaram o trabalho. O facto de o comportamento não ser o reportado por algumas das empresas do setor, levou-nos a questionar as atualizações dos sistemas realizadas depois do conhecimento destas ameaças. Isto dificultou o processo de análise da mesma, originando algumas das ideias para possível trabalho futuro. De qualquer forma, o objetivo de concretizar e documentar uma análise, a pelo menos uma amostra de *malware* moderna, foi concluído.

A primeira sugestão de trabalho futuro aponta para a realização de uma análise avançada estática mais detalhada a qualquer uma das amostras. Outros pontos que se consideram importantes para desenvolvimento futuro são: (i) a realização de um estudo e demonstração semelhante para *malware* direcionado a outros sistemas operativos em voga; (ii) a criação de uma plataforma de partilha de informação e ficheiros gerida maioritariamente pelas empresas da especialidade; (iii) a criação de um repositório de amostras de *malware* já catalogadas e identificadas; e (iv), criação de uma ferramenta *standalone* para análise básica automatizada. Outro ponto focado no trabalho futuro centra-se na implementação de uma interface gráfica para facilitar a interação com algumas das aplicações utilizadas e que só disponibilizam uma interface de linha de comandos.

Definições Úteis, Vinculação de Bibliotecas e Registo do Sistema

No anexo A surgem algumas definições importantes para a compreensão dos assuntos abordados no corpo desta dissertação. Aqui, são expostos os assuntos de empacotamento e ofuscação de *malware*. Da mesma forma, surge a explicação detalhada do termo *Strings* e *Portable Executable File Format*. Os outros assuntos explorados neste anexo são a vinculação de bibliotecas de *software* e o registo de sistema dos sistemas operativos da família Windows.

Análise Básica Estática ao `Tinba/Zusy`

O anexo B documenta a análise básica estática realizada à amostra do `Trojan Tinba/Zusy`. Para isso, apresenta toda a informação obtida através do recurso às diversas técnicas de análise básica estática.

Abstract

The exponential growth of the number of malicious software samples, known by *malware* in the specialized literature, constitutes nowadays one of the major concerns of cyber-security professionals. The objectives of the creators of this type of malware are varied, and the means used to achieve them are getting increasingly sophisticated. The increase of the computation and storage resources, as well as the globalization have been contributing to this growth, and fueling an entire industry dedicated to developing, selling and improving systems or solutions for securing, recovering, mitigating and preventing malware related incidents. The success of these systems typically depends of detailed analysis, often performed by humans, of malware samples captured in the wild. This analysis includes the search for patterns or anomalous behaviors that may be used as signatures to identify or counter-attack these threats.

This Master of Science (Ms.C.) dissertation addresses problems related with dissecting and analyzing malware. The main objectives of the underlying work were to study and understand the techniques used by this type of software nowadays, as well as the methods that are used by specialists on that analysis, so as to conduct a detailed investigation and produce structured documentation for at least one modern malware sample. The work was mostly focused in malware developed for the Operating Systems (OSs) of the Microsoft Windows family for desktops. After a brief study of the state of the art, the dissertation presents the classifications applied to malware, which can be found in the technical literature on the area, elaborated mainly by an industry community or seller of a security product. The structuring of the categories is nonetheless the result of an effort to unify or complete different classifications. The families of some of the most popular or detected malware samples are also presented herein, initially in a tabular form and, subsequently, via a genealogical tree, with some of the variants of each previously described family. This tree provides an interesting perspective over malware and is one of the contributions of this programme.

Within the context of the description of functionalities and behavior of malware, some advanced techniques, with which modern specimens of this type of software are equipped to ease their propagation and execution, while hindering their detection, are then discussed with more detail. The discussion evolves to the presentation of the concepts related to the detection and defense against modern malware, along with a small introduction to the main subject of this work. The analysis and dissection of two samples of malware is then the subject of the final chapters of the dissertation. A basic static analysis is performed to the malware known as *Stuxnet*, while the Trojan Banker known as *Tinba/zuzy* is subdued to both basic and advanced dynamic analysis. The results of this part of the work emphasize difficulties associated with these tasks and the sophistication and dangerous level of samples under investigation.

Keywords

Static Analysis, Dynamic Analysis, Dissection, Malware, Malicious Software, Stuxnet, Tinba/Zusy, Trojan, Computer Virus, Worm, Antivirus, Malware Families, Malware behavior.

Contents

List of Figures

Acronyms and Abbreviations

ACM Association for Computing Machinery

AI Artificial Intelligence

a.k.a. also known as

AOL America Online

API Application Programming Interface

ARPANET Advanced Research Projects Agency Network

ASCII American Standard Code for Information Interchange

AV Antivirus

BHO Browser Helper Object

BYOD Bring Your Own Device

CARO Computer Antivirus Research Organization

CD Compact Disk

CLI Command Line Interface

CP Cryptographic Provider

CPU Central Processing Unit

DDoS Distributed Denial of Service

DLL Dynamic Link Library

DNS Domain Name System

DoS Denial of Service

EXE Executable

FTP File Transfer Protocol

GINA Graphical Identification and Authentication

GUI Graphical User Interface

HIDS Host based Intrusion Detection System

HKCU Handle to Registry Key for the Current User

HTML HyperText Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

IAT Import Address Table

IB Instance Based

IBM International Business Machines Corporation

ICQ Internet Chat Query

IFrame Inline Frame

IDS Intrusion Detection System

IM Instant Messaging

IP Internet Protocol

IPS Intrusion Protection System

IRC Internet Relay Chat

IT Information Technology

MAC Media Access Control

MD5 Message-Digest algorithm 5

MIT Massachusetts Institute of Technology

Ms.C. Master of Science

MS-DOS/DOS Microsoft Disk Operating System

MSN Microsoft Network

MSP-CV Multimedia Signal Processing Covilhã Group

MS SQL Microsoft Sequel

NAT Network Address Translation

NIC Network Interface Card

NIDS Network based Intrusion Detection System

OS Operating System

OSI Open Systems Interconnection

PE Portable Executable

PEB Process Environment Block

php Hypertext Preprocessor

PID Process Identifier

PLC Programmable Logical Controller

POP3 Post Office Protocol 3

PSH Pass-the-Hash

PSW Password Stealing Ware

P2P Peer to Peer

RAM Random Access Memory

RAR Roshal ARchive

RAT Remote Administration Tool

RFID Radio-Frequency Identification

RSA Rivest, Shamir and Adleman

SHA-1 Secure Hash Algorithm 1

SMS Short Message System

SMTP Simple Mail Transfer Protocol

SVM Support Vector Machine

TCP Transmission Control Protocol

TCP/IP Transmission Control Protocol / Internet Protocol

TLS Transport Layer Security

URL Uniform Resource Locator

USA United States of America

USB Universal Serial Bus

VoIP Voice over Internet Protocol

WWW World Wide Web

XML eXtensible Markup Language

XOR Exclusive OR

Chapter 1

Introduction

This Master of Science (Ms.C.) dissertation is focused on modern malware analysis, a subject that is not thoroughly explored or that, sometimes, is surrounded by some mysticism. In this chapter, the motivation to approach this subject is discussed, as well as the adopted approach. It is also possible to find a brief history of malware, towards the end of this chapter. The last section contains a brief presentation of this Ms.C. dissertation structure.

1.1 Motivation and Scope

The history of computer malware can be traced back to 1962, where a group of Bell Labs designed a game that destroyed software programs. However, it was only in 1986 that the first computer virus for Microsoft Disk Operating System (MS-DOS/DOS) was reported. The virus was called Brain and it was the 1st virus at a computer running Windows. Brain propagated by Floppy Disks, and its only function was to infect the Boot sector and change the Floppy Disks Label [Lav13, Hyp11]. From this date on, the number of this kind of software and diversity of techniques used to propagate them kept growing [Mic], along with their purpose and number of platforms to create them.

Due to the growth of computation, and to the globalization of computers, it was just a matter of time until some of the major services and companies in the world, adopted computer systems, and this fostered malware developers. Malware stopped being the once simple software someone did to make a point, and became a serious threat, specially after attackers noticed that they could take profit out of it.

Nowadays, it is possible to find malware for all computer Operating Systems (OSs), and the most prolific system for malware writers is the Google Android OS, followed shortly thereafter by Microsoft Windows OS. Nowadays, the concept of Bring Your Own Device (BYOD) is one of the big reasons for malware to focus on Google Android OS. It becomes specially important to notice that Android is the most sold mobile OS worldwide and, due to the BYOD policy, it becomes easy to infect and obtain information from very important systems and organizations. On top of that, every smartphone easily connects to a lot of other smartphones, which makes them a perfect means of transportation for maliciously intended software. However, despite Google Android OS is on the vogue today, and being malware for Android OS a very highly profitable market now, this Ms.C. dissertation will focus on Microsoft Windows Malware, since it is still the most

widespread OS in the world. Malware is not a new topic, but it had a expressive evolution since the early stages of computing. Due to this evolution, there are a lot of different classifications of malware, which are used to distinguish the behaviour of the produced malware. Due to all this development, OSs developers where force to take additional measures and implement tools to provide security to their users. Users and Companies were also forced to somehow address the related security issues.

In order to provide the best security solutions to computer systems, it is required to study malware and find ways of analysing it. This was on the basis of the motivation for conducting this work. There is always the need to continuously analyse malware samples, since malware is constantly evolving to address modern detection techniques. Under the 2012 version of the Association for Computing Machinery (ACM) Computing Classification System, a *de facto* standard for computer science, the scope of the masters programme, reflected in this dissertation, falls within the categories named:

- **Security and privacy Malware and its mitigation,**
- *Security and privacy Systems security,*
- *Applied computing Computer forensics,* and
- *Social and professional topics Malware / spyware crime.*

1.2 Problem Statements and Objectives

The main problem addressed in this Ms.C. dissertation is the one of modern malware analysis. This problem is vast, and the work will be focused on a set of procedures to analyse modern samples for a single OS. Due to the motivations and to the character of the software involved, this type of task is normally less rich in terms of documentation and surrounded by some mysticism. New variants of malware are created each day, implementing novel obfuscation and propagation techniques. It is, as such, difficult to keep the pace with evolution on this area. Studying malware is, because of that, always a challenging task.

The objectives of this Ms.C. programme are to study and document particular samples of modern malware. As secondary objectives, it is intended to obtain new knowledge on the most recent technologies applied by malware developers. Additionally, the creation of a malware family tree, representing a snapshot of some of the current or historically worst menaces is also an objective of this work. Finally, it is intended to also conclude with some remarks that may aid practitioners on this area to conduct their analysis and to potentially point out guidelines for devising malware detection mechanisms.

1.3 Adopted Approach for Solving the Problem

In order to address the problem at hands, the first task was to perform a study on the areas considered relevant for this work. This task comprised an in-depth study of scientific and non-scientific articles and reports. Following that task, a report with the state of the art was produced, aiming to contextualize the topic and structure the remaining tasks. Following this report, a malware family tree was produced, so as to obtain a general perspective over the terms and some malware variants. In order to provide an accurate idea of the analysis process, it was envisioned that one or more malware samples would be analysed and documented in this dissertation and technical reports, comprising tasks of this work. Prior to these tasks, it was planned to study and assemble a safe environment for conducting the experiments. The analysis was to be divided into two main phases, namely, the basic and advanced analysis.

Though the approach was structured as discussed above, the last phase of the last task was to be approached on a best effort basis, meaning that, given the extension and nature of the work, it could not be possible to tackle of the topics. Since this work deals with malware analysis, prone to several problems, this approach seemed to be adequate.

1.4 Main Contributions

The main contributions of this masters programme and dissertation are as follows:

1. a review of the state of the art in terms of malware nomenclature was performed and structured in chapter 2;

2. family trees for 24 known malwares, built in accordance with a naming convention used by the industry, is delivered in chapter 3;

3. a way of setting-up a safe environment for malware analysis, is described in chapter 5; and

4. a detailed static and dynamic analysis of modern malware samples was performed during the programme and reported in chapters 5 and 6.

To the best of the knowledge of the author, there is no other efforts in producing (graphical) family trees like the ones resulting from this work. Malwares included in those trees are both modern and classical, but they were chosen with basis on popularity. In other words, older malwares are included because their samples are still in the top 10 most detected threats of some anti-malware solutions. Though of secondary importance, this work comprises an attempt to dwelve into a topic where up-to-date documentation is scarce, namely of malware samples, and sometimes surrounded by some mysticism. The work describes several steps that a forensic

analyst performs when analysing modern malware, which may be useful for other similar academic or industrial works or specialists on the area. The malware sample(`Tinba/Zusy`) analysed in the scope of this programme is recent and there are no reports with the same level of detail of its analysis in the literature, comprising thus one of the contributions of this work.

1.5 Brief History of Malware Development

The problems related with malware are not recent. The first MS-DOS/DOS virus was reported in 1986 and the history of malicious software goes back to 1940. This section aims to contextualize the subject at hands by providing a summary of the history of malicious software, focusing on Microsoft related incidents. This section is mostly based on [Lav13, Hyp11].

Back in the 40s, a Hungarian-American Mathematician called John Von Neumann studied the self-reproducing mathematical automata. This automata idea was to determine how complex systems can be generated by a reduced set of simple rules and objects. In 1951, Neumann had demonstrated how to create such automata. In 1959, Lionel Penrose, a British mathematician described the Neumann idea as a simple two dimensional model of the original structure, which could be activated, multiplied, mutated and attacked. Shortly after Penrose launched is idea, Frederick G. Stahl reproduced it on an International Business Machines Corporation (IBM) 650 program. These mathematicians where not trying to provide the basis for the future malware creators, and they did not want to provide tools for the nowadays malicious users, however, indirectly, they did. In a certain way, these studies motivated robotics and Artificial Intelligence (AI) studies and important developments.

In 1962, a group of engineers from Bell Telephone Laboratories, V. Vyssotsky, G. McIlroy, and Robert Morris, created a game that consisted on a battle of computer programs. The game had a referee in the memory of the computer that determined the rules and order of the battle between the competing programs created by the players, and their main objective was to destroy the opponents program, and take control of the battle field by multiplying. This was like a demonstration of the self-multiplying theories. Once the world saw this, it realized that these theories could be used for different purposes.

In 1971, the Creeper virus appeared on Advanced Research Projects Agency Network (ARPANET), which was considered the alpha version of the modern Internet. This virus was written for the then-popular Tenex OS. Creeper virus was able to propagate itself to a remote system via modem. At this stage, malware was not truly dangerous. For example, the Creeper virus only displayed the message `I AM THE CREEPER: CATCH ME IF YOU CAN`, after infecting the victim.

Back in 1974, a virus named Rabbit appeared, whose name was due to its multiplication and spread skills. Once it multiplied to a certain level on the infected machine, the machine would

crash.

1981 was the year of the first appearance of a virus on the widespread used Apple II platform. Elk Cloner was its name, and it infected the boot sectors, generating messages and turning the system slower.

The buzzword *virus* was popularized on November 10, 1983, at a seminar on computer safety at Lehigh University. Leonard Adleman demonstrated a virus-like program that was able to install itself on other system objects. A year later, on 1984, he defined *computer virus* as a program which is able to infect other programs by modifying them with the purpose of installing copies of itself.

The Microsoft Windows era started in 1985, with the launch of Microsoft Windows 1.0, introducing the first Graphical User Interface (GUI) created by Microsoft. This was an evolution from the MS-DOS/DOS Command Line Interface (CLI) to the pointing and clicking with a mouse in windows, menus, scroll bars, icons and dialog boxes. After the launching of Windows 1.0, Brain appeared, being now considered the first computer virus for Microsoft Windows. It was made by two Pakistani brothers with the intent of proving that business and consumers where clueless about security on their country. This Pakistani experience soon got out of control, and the virus spread worldwide. Another interesting fact about Brain is that it was the first *stealth virus*. When an attempt to read the infected sector was detected, the virus would display the original, uninfected data.

1987 was another important year on the malware and anti-malware history. The Vienna virus appeared in this year, spread all over the world and destroyed data on the infected computers intentionally. There was a worldwide debate, and a global effort to discover the creators of the program. One of the potential authors of the Vienna virus, Bernt Fix, was able to neutralize the virus. Fix is considered a precursor of modern Antivirus (AV) professionals and malware analysts and the program he created to neutralize the virus is considered the predecessor of todays AVs.

The first computer worm (the Morris worm) appeared in 1988. It spread through the Internet and it was written by a student at Cornell University called Robert Tappan Morris and was launched from Massachusetts Institute of Technology (MIT).

In 1990, a virus capable to change and adapt itself to avoid detection appeared in the wild. This virus was the first documented polymorphic virus and, at the time was simply called Chameleon. One year later, in 1991, the number of different malware samples was around 300 and several AVs vendors entered the market trying to provide anti-malware services to computer users. In the same year, the World Wide Web (WWW) was introduced by Sir Tim Berners-Lee.

Virus developers started to seriously focus their attention on the Microsoft Windows OS in 1996.

The appearance of the first virus for Windows 95, the Boza virus, is a proof of that. At the end of the same year, Laroux, the first Excel virus, is sighted in the wild.

In 1998, viruses like DeTroie appeared. This virus family, infected Win32 Executable (EXE) files and was able to transmit information about the victim machines to the malware creator. However, these viruses were only capable of working and spreading on the French version of Windows, due to some Application Programming Interfaces (APIs) used. As such, this epidemic affected only the French-speaking countries.

Back in 2001, e-mail and the Internet became the primary propagation means of malware, which resorted to scripts that would automatically load viruses from infected websites. A lot of malicious programs that exploited vulnerabilities in applications and OSs where launched in 2001, namely CodeRed, Nimda, Aliz and Badtransall. The large scale of these epidemics, specially the CodeRed and Nimda, setted the pace and trends for malware evolution for several years to come.

In 2003, another mark on the malware history, two global Internet attacks took place and, till today, they are recognized amongst the biggest in the history of the Internet. Slammer used an Microsoft Sequel (MS SQL) Server vulnerability. In a few minutes, the worm infected hundreds of thousands of computers through the world, and increased network traffic to such a point that several nations segments of the Internet crashed. The worm attacked computers through ports 1433 and 1434 and remained in computer memory. The second more important epidemic was caused by the Lovesan worm. 2003 is also the year that sees the first proved money making virus created, Fizzer.

In 2005, Sony launches a secret Trojan rootkit technology on Compact Disks (CDs). This rootkit technology was used to protect the CDs from illegal copying. However, the very same technologies could equally be used for criminal purposes, and that is exactly what happened almost immediately thereafter. Other rootkits appeared, providing hidden access to systems. In the history of malware, 2007 is mostly known has the year of the Botnet infections, the mass mailing of spam, Denial of Service (DoS) attacks and the compromising of passwords and data.

In 2009, GPCode introduces the term *Ransomware*. This malicious software awaits for the computer to be idle and then encrypts the desktop data and changes the desktop to a message demanding for a ransom. If the money is sent to a specific account, an actual password is sent to the affected user, so that (s)he can decrypt the data. Normally, these kind of malware will target corporate computers, since the loss of information on those computers may potentially have a higher value than on personal computer systems.

The `Stuxnet` legend starts in 2010. It silently ran in the wild for one entire year without being

noticed. This malware is undoubtedly one of the most sophisticated pieces of malware ever written. According to publications on the area, `Stuxnet` was created by the United States of America (USA) intelligence services with the specific intention of crashing the nuclear plant of Iran. This malware was transmited via Universal Serial Bus (USB) sticks and would only release its payload on a Siemens Programmable Logical Controller (PLC). After it was discovered, it was claimed that it already achieved its objectives. To halt the nuclear plant systems, `Stuxnet` would increase the centrifugation fans while showing a stable situation on the monitors, until the centrifuges crashed. At the time of the writing of this dissertation, it remains a mystery on how the piece of malware got into the PLC on the nuclear power plant, even though a lot of theories where drawn. It is said that `Stuxnet` took 10 years to program and it had only 1 MB of code. `Stuxnet` was nevertheless dethroned of the most evasive piece of malware by the Flame malware, which was supposedly hidden for 4 years, and only detected in 2012. According to experts, Flame and `Stuxnet` are the most sophisticated malwares ever written and spread worldwide.

Back in 2013, CryptoLocker, a Ransomware application whith several propagation techniques was the motive of concern for many experts. One of the propagation techniques was for the program to disguise as a legitimate attachment to a spam email. Once activated, CryptoLocker encrypts all the files of some specific extensions, on the local and Network connected drives, with an Rivest, Shamir and Adleman (RSA) public key. The private key is then stored on the CryptoLocker website, and is only available through a limited period of time. During that period, a payment is demanded, either via BitCoin (which is untraceable) or via a pre-paid voucher. Either way, the hacker is not detected, and cannot be linked to the payment. After the payment, the user gets the information and the private key needed to decrypt the encrypted data. However, it has been reported that sometimes files were not decrypted after the payment.

Computer systems face a lot of threats nowadays, comprised by a lot of different malware techniques and types. According to Kaspersky Lab, there are 315000 new pieces of malware appearing daily [Kas13]. It is clear that malware is a big threat on computer safety worldwide. One of the major concerns now is related with mobile solutions, which are gaining importance for normal or corporate users, e.g. through the concept of BYOD.

1.6 Dissertation Overview

This dissertation is organized in seven chapters, and their contents can be summarized as follows:

- Chapter 1 - **Introduction** - includes a brief introduction to the subject of this Ms.C. dissertation, as well as the discrimination of the motivation and scope of this work. The chapter is concluded with a brief history of Malware, to contextualize the subject at hands.

- Chapter 2 - **Background** - starts with a brief history on related scientific works and surveys and then elaborates on the classifications of malware samples, providing also the jargon and some background on the area. This classification is based on the values that samples are assigned to in modern anti-malware solutions.

- Chapter 3 - **Malware Families and Information** - provides a study on malware from a more practical and statistical point of view, and starts by explaining a malware naming convention. After introducing how anti-malware solutions name and categorize threats, the ones corresponding to the most detected threats by anti-malware solutions are exhibited and described. These threats lead to the construction of the malware family variants tree, included in the chapter. Finally, the chapter ends by providing a few technically detailed information on malware behaviours, hiding, data-encoding, anti-virtual machine, anti-debugging and anti-disassembly techniques.

- Chapter 4 - **Fighting Malware** - resumes the study of the state of the art, focusing on Anti-Malware techniques, tools, and behaviours. On this chapter a few detailed security tips are given, followed by the explanation of all types of security solutions available. Towards the end, the process of malware analysis is introduced with a high-level of abstraction.

- Chapter 5 - **Basic Static and Dynamic Analysis** - completes the introduction to the analysis matter, performed on the previous chapter, with a detailed analysis performed to a known malware sample.

- Chapter 6 - **Advanced Analysis** - continues describing the analysis procedure, started in the previous chapters, this time by reporting the advanced analysis to `Tinba/Zusy` malware sample.

- Chapter 7 - **Conclusions and Future Work** - concludes this dissertation by drawing the main conclusions of this Ms.C. programme, and by presenting a few guidelines for future work.

This dissertation contains some annexes too. There contents can be summarized as follows:

- Appendix A - **Useful Definitions, Library Linking and Windows Registry** - defines packing, obfuscation and strings, discusses library linking and the format of a Portable Executable (PE) File. It contains also a discussion on the Microsoft Windows OS Registry.

- Appendix B - `Tinba/Zusy` **Basic Static Analysis** - contains a discussion on the basic static analysis of `Tinba/Zusy`, since the corresponding chapter on the dissertation refers to `Stuxnet`.

Chapter 2

Background

2.1 Introduction

Malware is a short word for *Malicious Software* and it is used to describe software intended to change normal computer operation, gather sensitive information, or to get the credentials required to gain access to private computer systems. It is popularly known as a computer program that inflicts damage to the user, company or network in which the system operates, and it can can appear in the form of binary (compiled) code, script or other software that can perform intrusive and/or hostile functions on an OS [Acu12, Wik14c]. Since its appearance, malware has evolved so much and its functionalities became so diverse that some classification systems just for this type of software emerged. In this chapter, the different categories for malicious software are introduced. Most of the discussion is based on the Kaspersky Lab. classification [ZAO14e, ZAO14h], since this laboratory is highly regarded on this specific area of the industry and knowledge. When needed, additional references are appropriately added to the discussion to signal some adaptations to the classification structure. The next section provides an overview over some related works and surveys to contextualize the classification presented afterwards. The subsequent three sections elaborate on the aforementioned classification, while an alternative categorization is briefly discussed in the next-to-last section, before the conclusions.

2.2 Overview of Related Works and Surveys

In this section, some publications related to malware classification and analysis are introduced and summarized. It is possible to find a very significant number of publications on this subject on scientific databases. Nonetheless, none of them presents all the subjects that are going to be introduced and discussed in this dissertation. Notice that it was decided to include this section in this chapter because, that way, chapter 2 could be immediately focused on describing malware classifications.

In [GKAB09], Gupta *et al.* performed an in depth analysis of a Malware dataset produced by Mcafee with 19 years of analysis, using Data Mining algorithms, in order to find patterns on malware families so as to find relations between them. They concluded that, on the dataset used, there where 669 distinct malware families, and that some of them had more than 50 members/variants. They claim that the technique they present is able to help experts to develop

new strategies to counter some attacks. They also claim that they can identify similar instances of malware, even when crossing different repositories with different naming schemes.

Thomas M. Chen and Jean-Marc Robert define viruses and worms as software able to self-replicate in [CmR04]. However, they use the classical definitions to distinguish them. A Virus, as defined in 1983 by Fred Cohen is a kind of software that is able to self-replicate, but in order to perform that operation, it needs to attach itself to another *host* program or document. When executed, the *host* launches the malicious payload on the system of the victim. A Worm is simply a software that is able to self-replicate without the need of a *host* program or document. In order to self-replicate, a Worm only needs an active network connection and system vulnerabilities. On this report, the authors consider the evolution of Viruses and Worms to be held in four different waves, and they describe this evolution in terms of objectives, techniques and motivations. After the study of Viruses and Worms, the authors concluded that this kind of software are here to stay and to get better in the future, specially due to the fact that the time to exploit a new vulnerability is decreasing over the years. As Worms are constantly looking for new infection vectors, it is important to notice that a worm outbreak might be possible on a short term, and so it would be important to have a global coordinated antivirus system.

In the work [WN07], Brian Witten and Carey Nachenberg gave a brief presentation of the threats and Countermeasures that where known and followed back in 2005. Though it was written approximately 10 years ago, it is still a very important and useful source of information, specially in terms of threat evolution. Even despite being a 2005 document, we can still see that these programs evolved a lot since the beginning until then, and it is possible to conclude that in the last decade, with the growth of computation and the growth of Internet and Internet relaying services, these types of software evolved even more. Their discussion is presented in terms of Purpose, Stealthiness and CounterMeasures. The Countermeasures evolution is the most important and, as such, the mostly focused subject of this report. They introduce some countermeasure techniques that are still used nowadays.

The objective of Jian Ming, in [Min], was to provide information on malware static and dynamic analysis techniques. His main points where to present the techniques and limitations, some existing obfuscation techniques and cultivate future progress on this area of research. According to Ming, there are 5 types of static analysis, namely: the string signature, the Byte N-grams and Entropy, the Syntactic Library Call, the Control Flow Graph and Semantics-awareness techniques. These techniques are useful, but in certain cases, they are not enough to obtain good results, specially due to the difficulty of analysing programming code made by others, or in binary form, which is typically the case of malware. When in binary form, the program needs to be disassembled and the assembly code analysed, which is time consuming and cumbersome. On top of that, malware can also be obfuscated, which makes it even more complicated to decompile or study. As for dynamic techniques, Ming divides them in 5: Monitoring the func-

tion calls, analyse the function parameters, track the information flow, track modifications to the system registry and the auto-starting extension points. As dynamic analysis techniques only analyse code when it is running, they become immune to obfuscation techniques. Ming also introduced some techniques that malware use to avoid being detected, which he names *anti-malware analysis techniques*, in which he defines 4 classes: Environment-sensitive malware; Illusion; Mimicry attacks; and Dependence Replacement Attack. He then concluded that dynamic analysis has more advantages over static analysis, specially due to the code obfuscation, and to the static analysis limitations. Therefore, recent studies are specifically focused on the dynamic analysis Techniques.

[Sha13] is a survey written by Vivek Vimal Shah with the main objective of providing some guidelines and information to Information Technology (IT) professionals working on malware analysis. He starts by defining the most common types of malware, defined as: Viruses, Worms, Trojan-Horses, Bots, Spywares and Rootkits. The next point covered is the malware classification mechanisms, which are based on machine learning algorithms, namely, the Instance Based (IB) learner, Naive Bayes, Support Vector Machine (SVM), Decision Tree and Boosted Classifier. According to Vivek, another important thing to describe is the existing Infection Vectors, which can be classified in three branches: Network, Drive-by Downloads and Social Engineering. Next, the Analysis Techniques are described, beginning with the static analysis techniques and ending with the dynamic analysis. The analysis techniques where based on the previously mentioned article [Min]. He also introduced some of the most important and commonly used tools to analyse malware samples. In conclusion, he claims that, before fighting malware, it is very important to familiarize with and prepare to the task at hands, by gathering and reading appropriate documentation. He also claims that this is a continuous process, given the fact that malware is constantly evolving and using different things to avoid detection.

[KM13] is a survey performed by Kirti Mathur and Saroj Hiranwal with the objective of providing information on the existing analysis techniques for obfuscated malware. They start by defining and presenting the most common types of malware. They explain how to identify malware executables, mainly with the usage of a detector, which can work using Signature-Based or Behaviour-Based techniques. After detection, the malware sample can be analysed. According to the authors, the analysis can be divided into three branches: dynamic, static and hybrid, where dynamic involves running the executable, static involves studying the malware without running its executable, and hybrid is a combination of both, arguing that it may be more effective than the other two. They also introduce some techniques used by malware for avoiding detection and analysis, like code obfuscation and code packing (see annex A). They mention the existance of some techniques for deobfuscation and unpacking, which can be of help in detecting and analysing disguised malware executables. The authors conclude that static analysis is better than dynamic analysis, mostly because the latter makes it hard to analyse a multipath

malware and has a significant performance overhead.

In [DS13], G. Padmavathi and S.Divya focused specially on Wireless networks threats. Wireless networks threats are included in one of the following five types of security threads: Authentication Threats, Accidental Fault on Systems, DoS Attacks, Threats on Adhoc Networks, or Malicious Propagation. The authors elaborate on a malware classification, so that the problems can be contextualized. For Padmavathi and Divya, the main wireless network attacks are associated to Viruses, Worms, Blended Attacks, Keyloggers and Rootkits. The authors then introduce some of the most important and exploited vulnerabilities, namely Buffer Overflow, File based, OS ones, Buffer-Overrun, Browser ones and Weak Access Control and some malware analysis techniques, like heuristic-based and signature-based. They both conclude that the most important points on Analysis and Detection techniques are the number of false-positives and the infection ratio.

A survey on malware detection mechanisms, namely resorting to the usage of reverse engineering is included in [PV]. Authors define the most common types of malware as being Viruses, Worms, Trojans, Spyware, Adware and BotNets. They introduced the concept of *malware detector*, which is a function for analyzing files and classifying them as malicious or non malicious. They provide an explanation on signature-based, specification-based and behaviour-based detection techniques, focusing more on the signature-based ones, explaining how they are performed when different kinds of malware (basic, polymorphic, metamorphic) are analyses. On this survey, some of the most common obfuscation techniques are introduced, namely: Dead-code-insertion, Code Transportation, Register Renaming and Instruction Substitution. One of the last points in the reference concerns the concept of *Similarity verifying*. Similarity Verifying is a technique, to find if the program is malicious or not, based on euclidean distance between one original version of a malware sample and a suspicious program. To apply it, the malware sample is divided into some parts along with the suspicious program. The key is then to use a method of calculating the similarity value between the suspicious program part and the malware sample part. A low level of similarity indicates the suspicious program is not malicious. Otherwise, it is classified as malicious. However, the similarity threshold that is used to decide whether the suspicious program part is malicious or not must be very well defined. If not, the number of false positives/false negatives will be very big, turning the mechanism not usable at all. Towards the end, the authors conclude that the development of malware is now justifying a good investment in the research and development of better disassemblers and further improvements on similarity analysis algorithms.

An up-to-date review of the state of the art on malware and its detection is delivered in [SSA13]. Saeed *et al*. gave a significant contribute to researchers and academics on this topic, since they compile malware related information regarding creation techniques, execution environments, propagation media and negative impacts on the image of victims. They also discuss malware detection techniques and technologies, offering and interesting comparison, specially in the

form of two small tables, which provide a very brief and concise summary to readers.

The survey in [RRC11], by Rehman *et al.*, starts by contextualizing malware development and defining the most common types of malware. It ends with the discussion of trends in malware designs and some of the latest attack models and mitigation strategies. The authors claim that the biggest risk to a network are still the users, which connect to websites that are piled with malware. They add that the most used technique to detect malware is the signature-based one however, with the appearance of polymorphic viruses, new techniques must be further developed and tested. It becomes important to study the new trends on malware development in order to be able to anticipate developers and fight new potentially dangerous malwares. Along with the new trends, it is crucial to study the new attack vectors, which may take malware to new environments, namely: social networks, Botnets, Voice over Internet Protocol (VoIP) attacks and Pay-Per-Click-Hijacking. Concluding, the authors claim that a company should implement a multi-layered defense strategy in order to protect the organization network from sophisticated threats.

Finally, in [ICP], Chionis *et al.* introduce and define malware, evolving to the discussion of the most common Antivirus techniques. Most Antivirus work by using static analysis techniques based on signature databases. This survey tries to emphasize some alternatives that can be used to perform malware analysis when no signature is available. One of the big challenges to the signature-based antivirus are the polymorphic virus, which can change some of their characteristics and behaviours in order to fool these type of systems. So, it becomes crucial to create a system that can flag this type of software as malicious. Authors actually tried to gather and combine several related procedures and derive a general approach to malware analysis. One of the parts of the survey focus on explaining the usual process of analysis of a polymorphic malware, concluding that even though their proposal is on a preliminary state, it is going to be further developed.

The Kaspersky Lab. classification is structured into 2 main classes at the top. Each one of following subsections is devoted to the definition of those two classes, and lists their several specific programs or tools.

2.3 Classification of Malware - Malicious Tools

Malicious tools are not malware programs. Nonetheless, some anti-malware companies treat them like malicious programs, because they comprise tools designed to create malware or perform malicious activities automatically or by demand. Unlike viruses, worms and Trojans, malicious tools do not present a direct threat to the computer they are running in, and the malicious operations of the program are only triggered by direct order of the user/attacker [ZAO14c].

The *Malicious Tools* class is typically subdivided into 10 different subclasses according to the behaviour or functionality of the program:

Constructor Constructor tools are used to create new worms, viruses and Trojans. These programs generate the source code of malicious programs, object modules, and/or malicious files. There are a few Constructors which have a Windows-type interface, in which a menu is used to select the type of malicious program, self-encryption settings, anti-debugging features, etc.

DoS - DoS programs are specifically designed to perform DoS attacks on the computer of the victim. These attacks typically consist of sending numerous requests to a remote computer that provides a service and, if that computer does not have the sufficient resources to process all of the requests, it will enter into a DoS state.

Spoofer - Spoofer programs change the address (e.g., the Internet Protocol (IP) or e-mail address) of the party sending a message or of the network requests. These programs may be used for a wide variety of purposes, namely to prevent the recipient of a message from identifying the true originating party or if one wants to retransmit a message sent by another party.

E-mail Flooder - E-mail Flooder programs are designed to flood email channels with typically useless (from the point of view of the receiver) and sometimes meaningless messages. These tools are sometimes used by spammers.

SMS-Flooder - Short Message System (SMS) Flooder programs are used to flood text message channels with typically useless (from the point of view of the receiver) messages. These tools are also used sometimes by spammers.

IM-Flooder - Instant Messaging (IM)-Flooder programs are designed to flood IM channels (like Internet Chat Query (ICQ), Microsoft Network (MSN) Messenger, America Online (AOL) Instant Messenger, Yahoo Pager, Skype, etc.) with messages, sometimes meaningless or useless. These tools are most of the times used by spammers also.

Flooder - Flooder programs are used to flood network channels (other than e-mail, instant messengers, and text messages, e.g., Internet Relay Chat (IRC)) with messages. These tools can also be used by spammers or, e.g., virus developers.

Hoax - Programs classified as Hoax do not directly inflict any damage on the computer of the victim. They usually just send messages advertising that something is wrong or that problems on the system are eminent, or warning the user of a threat that does not actually exist. The objective of Hoaxes is to frighten users with messages, e.g., concerning refor-

matting their hard drive (although formatting is not actually being done), and displaying typical messages of viruses, etc., depending sometimes on the *sense of humour* or *imagination* of their creator. The final objective is to lead the user to install a truly malicious program or to ask him for money.

VirTool - VirTool programs can be used to modify other malicious programs so that they cannot be detected by AV.

HackTool - HackTool programs are used to create new users in the list of authorized system visitors, and to change or delete information from system logs in order to hide the presence of malicious users on the system. These programs are also used to collect and analyse network packets to carry out specific malicious actions. Malicious users employ HackTool programs when setting up attacks on local or remote computers.

Backdoor - A backdoor is a typically small program that runs in the background and provides remote control over the infected computer to its creator or distributor, allowing the attacker to perform various actions on that computer. Such actions may include sending, receiving, executing and deleting files, displaying data and rebooting the computer. This type of programs(which often fall within the trojan category) is normally used to unite a group of victims to form the so called botnets, or zombie networks, which can then be used to other, sometimes criminal, purposes.

Exploit - An Exploit is a program that contains data, or executable code, which may take advantage of one or more bugs, security failures or glitches, of a local or remote computer, to automatically inject software for malicious purposes. Malicious users usually employ an exploit to penetrate the computer of the victim and subsequently install malicious programs. Additionally, exploits are commonly used by Net-Worms, in order to infect systems without the authorization of the users. Nuker programs are notable amongst exploits. Such programs send specially crafted requests to local or remote computers, causing the system to crash.

2.4 Classification of Malware - Malicious Programs

Malware is typically subdivided into two main types, in accordance with the target of the attack:

1. Mass Malware, which refers to malicious programs that affect as many machines as they can; and

2. Targeted Malware, which refers to malicious programs that affect only a specific group, organization or user.

Apart from these two types, related with the target, malware can also be classified in terms of the actions it performs when it gets to the targeted computer or network. Regarding their actions, it is possible to isolate 4 categories of malware [ZAO14d], described in the next sections.

2.4.1 Rootkit

Rootkits are designed to hide certain objects or activities in the system. They are often used to prevent malicious programs from being detected on the system and thus grant that the malicious programs which are associated with it, remain undetected and running on the infected computer for the longest period of time. The objects and activities that this programs are designed to conceal include the Registry Keys (per instance, those used to automatically launch malicious objects), files, folders, and processes in the memory of the infected computer, as well as malicious network activity.

2.4.2 Trojan Programs

Trojans, unlike malicious tools, are malware that perform actions without the permission of the user. These actions may include manipulation of data, and disruption of the performance of computers or computer networks. Unlike computer viruses and worms, trojans are not able to self-replicate. In other words, they cannot make copies of themselves and propagate through the local or remote network by themselves. Trojans can be sub-classified according to the actions that they perform on the victims computer [ZAO14i]. This classification is as follows:

Trojan-Downloader - Trojan-Downloaders can download and install new versions of malicious programs into a computer, including trojans and adware. Once the download is complete, the programs are then launched or included in the system boot so that they can run automatically when the OS boots up. Information on the names and locations of the programs that the trojan might download are present on its source code, or the list is downloaded from an Internet resource, usually from a web page. This type of trojan is often used in the initial phase of infection, in which the victim visits websites that contain exploits.

Trojan-Dropper - Trojan-Dropper programs are designed to secretly install malicious programs built into their own code on targeted computers. This type of trojan usually saves a set of files to the hard drive of the victim (e.g. to the Windows directory, the Windows system directory, temporary directory, etc.). Those files are then launched without any notification, or with a fake one. Such programs are used to:

- secretly install trojan programs or viruses; and
- protect known malicious programs from being detected by AV solutions (not all AVs are capable of scanning all the components inside this type of trojans).

Trojan-PSW - Trojan-PSW are programs designed to steal passwords and other confidential data without using keystroke logging. A Password Stealing Ware (PSW) trojan searches system files or the registry in order to get the data. If such data is found, the PSW program sends it to the attacker. PSW trojans send the data using e-mail, File Transfer Protocol (FTP) or Hypertext Transfer Protocol (HTTP) (including the data in a request) however, there are other means by which stolen data can be transferred. These trojans can also steal registration information for some software.

Trojan-Spy - Trojan-Spy programs are used to spy the system of the victim. The so-called master of the spy program can see the data inserted via keyboard, make screen shots, see which applications are running, etc. The harvested information, like on Trojan-PSW, is sent to the master by using any of the data transfer means available.

Trojan-DDoS - Trojan-DDoS programs are designed to conduct a DoS attack from an infected computer. However, this type of trojan, is normally not intended to affect only one machine. To conduct a successful DoS attack, it is nowadays common to have a Botnet or Zombie-Network performing the requests, so that the attack is magnified in such a way that the computer or server stops responding to legitimate requests. So, this trojan is usually behind the creation of Botnets.

Trojan-Ransom - This type of program is similar to a human kidnapping and ransom situation. Once the malicious program infects the system of the victim, it blocks the access to certain data or prevents the computer from running correctly. Once the data has been *taken hostage* (meaning that it is blocked or encrypted), the user will receive a ransom demand. The ransom demand usually tells the victim to send money to the malicious user. If the victim does this, the malicious user then sends a program, or key to the victim to restore the data, or restore the performance of the system.

Trojan-GameThief - This type of trojan is intended to steal user account information for online games. The data is transmitted to the malicious user controlling the trojan via e-mail, FTP, the HTTP or any other existing data transfer means.

Trojan-IM - This kind of trojans are designed to steal user authentication data (logins and passwords) for IM applications, such as ICQ, MSN Messenger, AOL Instant Messenger, Yahoo Pager, Skype, etc. As in other similar trojans, the data is then transmitted using any of the existing data transfer means to the malicious user controlling the Trojan-IM.

Trojan-Banker - Trojan-Banker programs are designed specifically to steal online banking systems credentials, e-payment systems and plastic card systems. The data is then sent to the malicious user controlling the trojan by one of the many existing data transfer means.

Trojan-Mailfinder - Trojan-Mailfinder programs are designed to collect email addresses from a computer and then send them to the malicious user controlling it, via any data transfer means available. These addresses are then used by the cyber criminal to conduct mass mailings of malware and spam.

Trojan-SMS - These trojan programs are usually found on mobile devices, and are often used to send text messages from the infected mobile device to premium rate numbers that are usually hard-coded on the source code of the trojan.

Trojan-Clicker - Programs classified as Trojan-Clicker are designed to access certain Internet resources (usually WWW pages). This is done by directly sending commands to the browser of the victim or by replacing system files that provide standard addresses for Internet resources (such as the `hosts` file on Windows OS). There are three main reasons for a malicious user to use this kind of programs:

1. increase the number of visits to certain sites, in order to boost the number of hits for online ads;

2. conduct a DoS attack on a particular server; and

3. directly lead potential victims towards viruses or trojans.

Trojan-Proxy - Malware classified as Trojan-Proxy is designed to give malicious users access to a variety of Internet resources via the computer of the victim. Trojan-proxies are often used by hackers to hide the location of the original host from any investigation authorities, as the connection can often only be traced back to the computer where the trojan is installed. Thus, this is the perfect tool to realize, for instance, a mass spam mailing scheme.

Trojan-Notifier - Trojan-Notifier programs are used to notify the malicious user controlling them when an infected computer is online. This notification can include information like IP addresses, the number of open ports, e-mail addresses, etc. Notifiers are embedded in generic trojans (trojans that fall into two or more subcategories) in order to notify the malicious users of the successful installation of malware on the computer of the victim.

Trojan-ArcBomb - This kind of trojans are specially crafted archives, engineered to freeze or decrease the performance of the hard drive with a large amount of empty data when an attempt is made to unpack them. The so-called archive bombs pose a particular threat for file and e-mail servers when the processing of incoming data is done automatically. An incoming archive bomb can simply crash the server. It is possible to isolate three types of *bombs*: (i) badly generated archive headers; (ii) repeating data; or (iii) identical files in the archive. Badly generated archive headers or corrupted data in an archive can cause

some packing or unpacking algorithms to crash, when processing the archive contents. As for the repeating data, packer algorithms make it possible to pack a large archive into a small one due to the repetition of data (e.g. 5 GB of data can be packed into a 200 KB Roshal ARchive (RAR) or a 480 KB ZIP archive). As for the identical files in an archive will also have little impact on the size of the archive when it is packed using special methods (e.g. It is possible to pack 10100 identical files into a 300 KB RAR or a 230 KB ZIP archive).

Trojan-FakeAV - Trojan-FakeAVs are programs that simulate an AV or some parts of the OS security modules. These programs are usually associated to Ransomware schemes. This malware usually shows repeating pop-ups in an effort to make the user worry about its system security, and pay for fake AV. Additionally, Trojan-FakeAV programs inhibit proper functions of the system, so as to make sure that the user believes that the threat is real.

For classification purposes, any malware presenting two or more of the above subclasses, is considered a General Trojan, not falling into any of the before-seen subclasses.

2.4.3 Viruses and Worms

Viruses and Worms are malicious programs that have the ability to self-replicate on computers or via computer networks without the user knowledge (though its interaction may be required - see below). Each subsequent replication of the Virus or worm is able to self-replicate as well. The viruses and worms subclass does not contain malicious programs that spread via networks or infect remote machines by command, or programs that create multiple copies that are then unable to self-replicate. The main feature used to determine whether or not a program is classified as a *Virus or Worm* is how the program propagates (i.e., how the malicious program spreads the copies of itself via local or network resources). One of the major differences between worms and viruses is that the former exploit bugs to automatically spread via computer networks and eventually replicate and infect machines. The latter usually needs the intervention of the user, which normally means that the program needs to be executed in order to infect the machine. When active, viruses spread mostly on local resources, tipically requiring the aid of the human to spread out. The most common techniques to spread worms are:

- files sent as e-mail attachments; and
- network packets.

Worms then use some techniques to penetrate the remote computers and launch copies of themselves, like social engineering (e.g., an e-mail suggesting the user to open an attached file), exploiting network configuration errors (such as copying to an accessible network drive), and exploiting loopholes in the OS and applications security.

As for viruses, they can be subdivided according to the method used to infect a computer. The subdivision is as follows:

- file viruses;
- boot sector viruses;
- macro viruses; and
- script viruses.

Any program falling within this subclass (viruses and worms) can also exhibit additional trojan functionalities. It should also be pointed out that many worms use more than one method to spread copies via networks. As such if any worm uses more than one method to spread copies of itself, it is considered a General worm, just like with trojans [ZAO].

This subclass of malicious programs can be structured according to two main behaviours, which are described in more detail in the following two subsections.

Worm

Worms use computer networks to spread out, exploiting bugs on intermediate or host devices. Their propagation is thus often faster than with viruses. The general *worm* category covers all the malicious programs fulfilling the conditions that define this class and that do not fit into any of the categories included below (e.g., worms for mobile devices). The main subcategories into which worms can be structured into are as follows:

IM-Worm - IM-Worms are malware that use any one of the IM systems to spread. They usually send an Uniform Resource Locator (URL) to a list of contacts, which leads to a network resource where a file containing the worm is placed. This tactic is very similar to the one used by E-mail Worms (see below).

P2P-Worm - P2P-Worms are spread via Peer to Peer (P2P) content sharing networks. In order to get into the P2P network, the worm has to copy itself to the file sharing directory, which is usually on a local machine. Then, the P2P system does the remaining work, i.e., in a P2P network, when a file search is performed, the remote users are informed of the existence of the file, making it available to download from the infected system. A second approach for P2P-Worms is to them to contain a version of the P2P client, initiating it on the infected machine and to respond positively to search queries, offering a copy of the P2P-Worm as a match.

IRC-Worm - This type of worm spread via IRC, using one of two basic approaches. The first one, just like IM-Worms, involves sending an URL that leads to a copy of the worm. The second one consists on sending an infected file to an IRC channel user. However, both techniques require the recipient user to accept the reception of the file, save and open it.

E-mail Worm - E-mail Worms are the ones that use e-mails to spread, using again one of two basic approaches. On the first case, the worm sends a copy of itself as an attachment to the e-mail. On the second case, the worm sends an e-mail with a URL to its copy on a network resource. On both cases, the worm requires the recipient to then open the file to be activated.

Net-Worm - Net-Worms are the ones that do not need any action from the user in order to spread. They usually search for critical vulnerabilities in software running on networked computers. In order to perform the infection, a specially created network packet (called an exploit) is sent , so that the worm code (or part of it) penetrates and gets active in the computer of the victim(s). Sometimes, the network packet only contains the part of the code needed to download and run a file containing the main module. Some Net-Worms use several exploits to simultaneously spread, thus making them more effective and fast on finding victims.

Virus

Virus replicate themselves on the resources of the local machine. Contrarily to worms, virus do not spread via network services. A copy of a virus will reach a remote computer only if the infected object is, for some reason unrelated to the virus function, activated on that machine. For instance:

- when infecting accessible network drives, a virus penetrates a file located on a network resource;
- a virus copies itself to a removable storage device or infects a file on a removable device; or
- a user sends an e-mail with an infected attachment.

2.4.4 Suspicious Packers

Malware samples are frequently compressed (or packed) using a variety of methods combined with file encryption, to prevent reverse engineering of the program and to difficult the analysis of program behaviours with proactive and heuristic methods. AVs have the ability to detect the

results of suspicious packers, i.e., they can detect packed items. There are a few ways to avoid packet files from being unpacked: for example, the packer may not fully decipher the code, deciphering only the part needed to execution; or it may fully decrypt and launch a malicious program only in a certain day of the week. The main features used to differentiate behaviours in the *suspicious packers* subclass are the type and number of packer used in the file compression process [ZAO14g]. This subclass can include the three behaviours described next:

MultiPacked - This class is applied to files that have been packed several times using a variety of packers. An AV will detect the executable file as MultiPacked if it is packed using three or more packers;

SuspiciousPacker - covers objects that have been compressed using packers designed specifically to protect malicious code against detection by AVs; and

RarePacker - covers files compressed by packers that are very rarely seen, for example proof of concept packers.

2.5 Classification of Malware - Adware, Pornware and Riskware

The class of Adware, Pornware and Riskware cover programs that are developed and distributed by legitimate companies, may have legitimate purposes, but they may include functions that, on certain conditions, can pose a threat to computer users. It is normal for administrators to install remote administration programs on computers they manage in order to remotely solve problems that may arise on the machine. However, when that same administration program is installed illegally on a computer by a malicious user, then the program falls in this category. As these programs are usually legitimate, in many cases, AVs cannot determine whether or not they pose a specific threat without input from the user [ZAO14a].

2.5.1 Adware

Programs belonging to the Adware class, are designed to display advertisements, redirect search requests to advertising websites and collect data concerning the preferences of the user (e.g., which types of websites he or she visits) in order to display customized advertisements on the computer of the victim. Apart from the actions mentioned before, these programs are not typically visible on the system. For instance, there will be no icon in the system tray, and no indication that the program has been installed. Often, this kind of software does not have any uninstall procedure and use technology similar to the ones of virus to help the program enter the computer undetected and run unnoticed. It is important not to confuse data collecting Adware with Trojan-Spy programs. The main difference is that Adware collects data with the consent of the user. If Adware does not notify the user that it is gathering information, then it is classified as a malicious program, specifically covered by the Trojan-Spy subclass. There are two main

ways that Adware uses to get into computers:

1. Built into some freeware and shareware programs, whose main purpose is to get a type of payment for the software by showing advertisements to the user. In this case, the parties who make the advertisements pay the advertising agency, and the advertising agency pays the Adware developer. This type of software also helps to cut expenses for software developers, mostly because the developers get a revenue (even if small).

2. Unauthorized installation to a computer as a result of a visit to an infected website, which uses exploits and other related technologies. For instance, a computer can be penetrated via a browser vulnerability and trojans (like Trojan-Downloader or Trojan-Dropper) can then install the Adware. Adware programs working like this are also known as Browser Hijackers.

Most of the freeware and shareware programs stop displaying advertisements when they have been purchased or registered. Often, these programs use built-in third party Adware utilities and, in some cases, these utilities remain installed in the computer even after the purchasing or registering of the software. In some cases, removing the Adware component can even cause the program to malfunction.

The two main ways in which advertisings are shown to the user are enumerated next:

1. By downloading advertising text and images to a computer from HTTP or FTP servers, owned by the advertiser;

2. By redirecting the Internet browser search requests to advertising websites.

2.5.2 Riskware

Riskware covers legitimate programs, that can cause damage when they fall into the hands of malicious users. Programs in this class include remote administration utilities, IRC clients, dialer programs, file downloaders, software for monitoring computer activity, password management utilities, and numerous Internet server services such as FTP, HTTP, proxy and telnet. These programs are not malicious by themselves, although they do have functions that can be used for malicious purposes. One example is the mIRC tool. Any IRC backdoor is capable of writing its own scripts to the mIRC configurations file and successfully deliver its malicious payload without the knowledge of the user. The mIRC user will not even suspect that a trojan is running on his computer. Often, malicious programs install the mIRC client themselves for communications. In such cases, mIRC is usually saved to the Windows folder and its subfolders. If mIRC is detected in these folders, it almost always means that the computer has been infected with some type of malware. This class can be further divided into the following 16 categories:

Client-IRC - Programs in this category are used to communicate using IRC and are not considered malicious programs. However, they are detected because malicious users frequently exploit their functionalities. Malicious programs can be used to install Client-IRC on the computer of the victim for malicious purposes.

Client-P2P - Client-P2P programs are used to interact with P2P networks and are not malicious programs. These programs were added to this list because some where the cause of leakage of confidential data in the past.

Client-SMTP - Client-SMTP is used to send e-mail and can run in an hidden mode. This type of programs is sometimes included in bundles of malware, by malicious users, so as to send mass spam mailings from the computer of the victim.

Dialer - The Dialer stealthily creates telephone connections via a modem.

Downloader - Programs of this type download a variety of content from network resources undetected.

FraudTool - Such programs frequently encourage users to transfer funds to a specific account in exchange for services that they may or not provide. On top of that, FraudTool programs, have several different naming, accordingly to the means they use to encourage people to believe in the fraud, namely scareware, which tries to encourage users by menacing them, and rogueware, that tries to fool the user into buying something that will help him turn his computer safer or faster. Most common Fraudtools are pseudo AV which display messages claiming that malware has been detected. These programs do not usually perform any of the advertised functionalities and are not able to solve infections.

Monitor - Programs of this category are able to monitor computer activity (active processes, network activity, etc.) and are not tipically considered as malicious programs.

NetTool - Programs classified as NetTools are designed to explore networking functionalities (for example, remotely rebooting a computer, scanning open network ports, remotely launching random applications, etc.).

PSWTool - Programs classified as PSWTool can be used to view or restore forgotten (often hidden) passwords.

Remote Administration Tool - Programs in this category are used to remotely manage a computer.

RiskTool - Risktools provide functions such as concealing files in the system, hiding application

windows, terminating active processes, etc. Unlike NetTools, RiskTools are designed to operate on the local computer.

Server-FTP - These programs function as FTP servers. For this reason, malicious users include them in bundles of malware in order to gain access to a computer that has the server installed.

Server-Proxy - Such programs function as a proxy server and are used to send out spam or other malicious content from the infected computer.

Server-Telnet and Server-Web - These programs are similar in purpose and functionality to the previous ones, but use telnet and HTTP as the communication protocols.

WebToolbar - Toolbars enhance the capabilities of user software and are installed with consent. However, some toolbars are installed along with other software components. These toolbars make use of special installers that employ a variety of methods to automatically receive installation permissions e.g., checking an *I agree* option by default.

If any of the defined Riskware programs (**except for FraudTool**) is installed purposely by the user, or if it was installed by a system administrator, then that software does not pose any threat. However, all of the above software types can pose a threat to the user if not used the right way.

2.5.3 PornWare

Pornware appears as the classification attributed to programs which display pornographic material to the user. As other programs in this category, Pornware may be installed with user consent, in order to obtain pornographic contents. In this case, it may not be unwanted, but it still can be considered malicious. However, they can be installed by malicious users, recurring to OS or browser vulnerabilities, or by specifically use trojans like Trojan-Downloaders and Trojan-Droppers. The purpose of this infection is to present the user with paid pornographic websites, that may not be of the awareness of the user. The behaviours of programs in this class define the three categories indicated below:

Porn-Dialer - These tools are used to dial to phone services or numbers related with adult contents. They can also be used to submit special promotion codes automatically and contained in their code. They usually notify the user of their actions.

Porn-Downloader - Tools of this type download pornographic related files to the computer of the user. They also usually notify the user about their activities.

Porn-Tool - Programs within this category are used to search and display pornographic contents

to the user (e.g.,they may be comprised by special toolbars for browsers and special video players).

2.6 Alternative Classifications

The number of malware variants increased drastically in the last few years. To ease the categorization of this type of programs, AV vendors created an alternative classification focused mainly on new trends in malware development. Because of that, *classes* are sometimes referred to as *trends* in this system. It groups all of the aforementioned classes into four specific categories [ZAO14b]. The most recent, persistent and dangerous samples usually fall into one of the newly defined classes. Each one of those classes may, nonetheless, cover several other classes discussed previously.

2.6.1 Crimeware

This specific category covers malware whose main objective is to commit financial crimes. Nowadays, there are hundreds of different programs which may fall under this specific trend. Programs that track banking and electronic payment systems are good examples of crimeware. Malware developers are constantly devising new schemes for making money with malware and, as such, this trend needs to be closely followed in the next few years. In the classification system presented in previous sections, crimeware samples would end up being a Trojan-Spy, Trojan-Banker, Trojan (Generic), Virus, IM-Worm, P2P-Worm, IRC-Worm, Worm(Generic), E-mail Worm or Net-Worm.

2.6.2 Spyware

Programs that are commonly used to spy or collect any information of the user, without his/her consent, are simply called Spyware in this classification. This trend includes keyloggers (programs that inform the attacker of the keys pressed by the victim in the computer), programs that send e-mail addresses from the computer of the victim towards the attacker, etc. In terms of behavior, malware samples on this trend are usually one of the following: Trojan-Spy, Trojan-PSW, Trojan-Notifier, Trojan-GameThief, Trojan-IM, Trojan-Mailfinder, Trojan-Banker, Trojan (Generic), Virus, Worm (Generic), IM-Worm, P2P-Worm, IRC-Worm, E-mail Worm or Net-Worm. A Trojan-Banker, previously classified as crimeware, can also be a Spyware, since it may collect information from the user with criminal purposes in both cases, turning it into one of the typical overlaps in this classification system.On the other hand, Adware should never fall into the spyware category because it does not perform any illegal action in the sense it requires the consent of the user. Nonetheless, there are many AV solutions that place them in this trend, mostly due to the fact that most of the users are not fully aware of the activities of such software, as a result of not reading the software disclaimers entirely.

2.6.3 Ransomware

A program is classified as Ransomware if it blocks the access or disrupt the performance of the computer in which it runs with the purpose of obtaining a payment for the attacker in return. These programs are sometimes equipped with means to detect the most valuable files and directories on the computer of the victim and most of the times their purpose is achieved via encryption of the referred resources. Nonetheless, it is also possible to find several examples of programs which can be considered Ransomware that resort to mechanisms to disrupt the computer performance instead of encryption. Usually, in both situations, a message is shown to the user, claiming for a ransom. Frequently, the malicious activities performed on the victim are reversed and the computer files and performance are restored after the ransom is paid. Usually, samples of Trojan-Ransom, Trojan (Generic), Virus, IM-Worm, P2P-Worm, IRC-Worm, Worm (Generic), E-mail Worm, and Net-Worm may all fall under this class.

2.6.4 Bot-Clients

The class of bot-clients is used to define all those programs used to construct botnets, which are nowadays used to generate spam mail and to perform Distributed Denial of Service (DDoS) attacks. Such malware sits on the system of victims, listening on a predefined port or any other communication channel previously arranged with the attacker, giving him or her control over the zombie machines. The programs that mostly fall into this category are the ones exhibiting backdoor behaviour. Nonetheless, it is possible that malware like Virus or Worms are classified as Bot-clients, since their set of features may include backdoors.

2.7 Conclusion

Classifying a malware sample constitutes an important objective of the malware analysis process. Throughout this chapter it was shown that this type of programs can be classified according to the payload, target or behaviour. The most recent classification system is more focused on trends, providing a simpler scheme containing 4 categories only. The number of classes discussed herein is a clear indicator of the complex habitat created by malware, which results in a very time consuming and intellectually demanding specialization area. In order to summarize the aforementioned classifications and all the discussion included in this chapter, table 2.1 and 2.2 where elaborated and included herein. They provide a structure perspective over both traditional and alternative classifications. Another important aspect of the malware analysis process is the name given to the samples, when detected by existing AV or anti-malware solutions. Because of that, the next chapter will be focused on naming conventions and on the families of some important malwares.

Table 2.1: Overview of the Traditional Classification System.

Traditional Classification				
Malicious Tools	Constructor		Adware	
	DoS		Riskware	Client-IRC
	Spoofer			Client-P2P
	Email-Flooder			Client-SMTP
	SMS-Flooder			Dialer
	IM-Flooder			Downloader
	Flooder			FraudTool
	Hoax			Monitor
	VirTool			NetTool
	HackTool			PSWTool
	Backdoor			Remote Administration Tool
	Exploit			RiskTool
Malicious Programs	Trojan	Trojan-Downloader		Server-FTP
		Trojan-Dropper		Server-Proxy
		Trojan-PSW		Server-Telnet and Server-Web
		Trojan-Spy		WebToolbar
		Trojan-DDoS	Pornware	Porn-Dialer
		Trojan-Ransom		Porn-Downloader
		Trojan-Gamethief		Porn-Tool
		Trojan-IM		
		Trojan-Banker		
		Trojan-Mailfinder		
		Trojan-SMS		
		Trojan-Clicker		
		Trojan-Proxy		
		Trojan-Notifier		
		Trojan-ArcBomb		
		Trojan-FakeAV		
	Worm	IM-Worm		
		P2P-Worm		
		IRC-Worm		
		Email-Worm		
		Net-Worm		
	Virus	File Virus		
		Boot Sector Virus		
		Macro Virus		
		Script Virus		
	Suspicious Packers	Multipacked		
		Suspicious Packer		
		RarePacker		

Table 2.2: Overview of the Alternative Classification System.

Alternative Classification			
Spyware	Trojan-Spy	Ransomware	Trojan-Ransom
	Trojan-PSW		Trojan (Generic)
	Trojan-Notifier		Virus
	Trojan-GameThief		IM-Worm
	Trojan-IM		P2P-Worm
	Trojan-Mailfinder		IRC-Worm
	Trojan-Banker		Worm (Generic)
	Trojan (Generic)		Email-Worm
	Virus		Net-Worm
	Worm (Generic)	Bot-Clients	Virus
	IM-Worm		Worm (Generic)
	P2P-Worm		IM-Worm
	IRC-Worm		P2P-Worm
	Email-Worm		IRC-Worm
	Net-Worm		Email-Worm
Crimeware	Trojan-Spy		Net-Worm
	Trojan-Banker		
	Trojan (Generic)		
	Virus		
	IM-Worm		
	P2P-Worm		
	IRC-Worm		
	Worm (Generic)		
	Email-Worm		
	Net-Worm		

Chapter 3

Malware Families and Information

3.1 Introduction

The identification of malware samples is a requirement since the beginning. With the growth that malware experienced in the last years, this requirement became even more significant. Nowadays, and according to the Kaspersky Lab, approximately 315000 new malware samples arise every day [Kas13]. The idea of naming malware is to cluster similar samples according to their behaviour, strings used in the code, etc. This name is then usually referred to as a new family of malware.

This chapter contains an introduction to the malware naming convention, as well as a discussion on the most widespread malware families nowadays. After this introductory section, the explanation focuses on malware functions and capabilities which produce some of the most noticeable behaviours, as well as on some of the techniques and information they can use to avoid being detected and analysed.

3.2 Naming Convention

There is a convention for structuring the name of malware, but no standard for the name per se. This means that the same malware may have different names in different AVs. The naming style convention that is respected by most of the AV vendors is called Computer Antivirus Research Organization (CARO) [CAR02, Bon, Cen14, ZAO14f]. On the CARO naming style convention, malware is named according to the following structure:

```
Behaviour:Platform/Name.Variant!Information
```

The `Behaviour` identifies what the detected threat does. On the case of viruses and worms, just like explained in Chapter 2, the behaviour reflects the propagation methods used. For trojans and malicious tools the behaviour is given by the malicious payload. On suspicious packers, it is given by the way they act. For the other types (like Adware, Riskware and Pornware), the behaviour is given by the function of the detected object. The `Platform` part of the name, expresses the environment on which the program code executes successfully. This can refer to software and hardware. The cases where the detected threats can be executed on more than one platform, the platform part of the name is defined as *Multi* (e.g. `Virus:Multi...`). The `Name` is the official name given to the detected threat, which also ends up defining a new

family of malware. The term family is used to describe a group of detected malware that shares the same origin (being either, the author, source code, operating principles or payload). The `Variant` part denotes a modification of a detected threat. The modification can be made by the author, or other developers. However, it may not alter the behaviour of the threat. The variant is usually indicated by a number or a letter. (e.g. *.A - .Z*, *.AA*). The variant field is not mandatory. The `Information` part is not mandatory either. However, it is used sometimes to provide information about a specific file or component that is used by another threat associated with this one. For example, *!lnk* indicates that the threat is a shortcut file used by another variant of the same or another family. Some AV companies, also add another part on the name, called *Prefix*. When present, this part is used in the beginning of the name, and usually takes one of two values: *HEUR* or *PDM*. On the *HEUR* case, the detected threat is detected by the heuristic engine, while on the *PDM* prefix, the threat is detected by the proactive defence module.

3.3 Industry Review

In the following subsections, statistics from some AVs vendors will be presented and discussed.

3.3.1 Top 10 Families by Microsoft for the 1st Semester 2013

Table 3.1 lists the top 10 malware and potentially unwanted software families that were detected on computers by Microsoft anti-malware products worldwide on the first semester of 2013 [Mic13].

Table 3.1: Top 10 families detected by Microsoft on the 1st Semester of 2013.

Order	Family
1	INF/Autorun
2	Win32/Obfuscator
3	HTML/IframeRef
4	JS/Seedabutor
5	Win32/Dorkbot
6	Win32/Sirefef
7	Win32/Sality
8	Win32/Conficker
9	Win32/Gamarue
10	JS/BlacoleRef

The families depicted in the table can be briefly summarized as follows:

`BlacoleRef` - On the last place, `JS/BlacoleRef` is an obfuscated script, often found inserted into compromised websites, that uses a hidden inline frame to redirect the browser to a blacole exploit server.

`GamaRue` - This malware family is a worm, sometimes associated to the blacole exploit kit, and distributed by exploit kits and social engineering or by other malware. Some variants of

this malware family are programs that can spread by infecting removable drives (like USB flash drives or portable hard drives). If the victim then plugs those drives into another computer, the worm will infect that computer as well. Variants for stealing information from local computers and communicating with Command-and-Control servers managed, by the attackers.

`Conficker` - This family of worms can disable several important Windows services and security products. They can also download files and run malicious code on the computer of the victim, if file sharing is enabled. `Conficker` worms infect computers across a network by exploiting a vulnerability in the Windows OS system file and forms a botnet while spreading.

`Sality` - The `Sality` family consists of polymorphic file infectors that target executable files with the extensions `.scr` or `.exe`. They may execute a malicious payload that deletes files with certain extensions and terminate security-related processes and services.

`Sirefef` - Programs of the `Sirefef` family usually disguise themselves as legitimate applications, such as utilities, games, or even free AVs. Attackers use this technique to trick the victims into downloading the fake application, and once the user allows the application to run on his or her computer, the hidden `Sirefef` malware is executed. The malicious payload of `Sirefef` includes performing click fraud and using the resources of the infected computers to *mine* for bitcoins.

`Dorkbot` - `Dorkbot` is a family of IRC-based worms that spreads via USB removable drives, IM programs, and social networks. Some `Dorkbot` variants may capture user names and passwords by spying on the network traffic of the victims, block websites that are related to important security updates and launch a limited DoS attack.

`Seedabutor` - `Seedabutor` has several variants that were first detected in the year of 2012, but where reclassified as a new family in January 2013. `Seedabutor` is intended to make the redirection of the browser of the user to specific websites, possibly controlled by the attacker.

`IframeRef` - `IframeRef` is a generic name for specially formed HyperText Markup Language (HTML) Inline Frame (IFrame) tags that redirect to remote websites that may contain malicious content.

`Obfuscator` - `Obfuscator` is a generic name for files or programs that have been modified by malware obfuscation tools and techniques. They usually use a combination of several encryption, compression, anti-debugging or anti-emulation techniques that alter the file or program in an effort to hold back the analysis or detection by anti-malware products. The files or programs obfuscated by these techniques usually have the same functionality

as the original non-obfuscated ones, but their code, data and geometry is different.

`Autorun` - `Autorun` is the most common threat found worldwide during the 1st Semester of 2013. This family name is used for the generic detection of worms that spread between mounted volumes using the `AutoRun` feature of Microsoft Windows OS. Changes to this feature on the newer OSs, like Windows 7 and Windows 8 made this technique less effective over time, but attackers continue to distribute malware that attempts to target it.

3.3.2 Top 10 Families by ESET on October 2013

Similarly to the previous subsection, table 3.2 presents the list of the top 10 malware and potentially unwanted software families that were detected by ESET cyber security solutions worldwide on October, 2013 [ESE13].

Table 3.2: Top 10 families detected by ESET on October 2013.

Order	Family
1	Win32/Bundpil
2	INF/Autorun
3	Win32/Sality
4	HTML/Iframe
5	HTML/ScrInject
6	Win32/Dorkbot
7	Win32/Conficker
8	Win32/Ramnit
9	Win32/Small
10	Win32/Qhost

The families presented in the previous table are briefly described below, but it should be noticed that `Win32/Conficker`, `Win32/Dorkbot`, `Win32/Sality` and `INF/Autorun` have already been explained on the previous subsection:

`Qhost` - `Qhost` is a family of trojans that change the contents of the `hosts` file of the victims to block the access to certain websites. It can also steal passwords by redirecting the details of the victim through an hacker server as specified on the changes to the aforementioned file.

`Small` - The variants of the `Small` family are often used to download and execute arbitrary files (including additional malware), chosen by the attacker, to an infected computer.

`Ramnit` - `Ramnit` family malware variants steal sensitive information from their victims like, for instance, bank user names and passwords. They may also provide access and control to the malicious user, as well as stop the security software from running on the computer of the victims. This malware usually spreads via USB removable devices.

`ScrInject` - `ScrInject` is a family of malware associated with the detection of HTML pages containing obfuscated scripts or IFrame tags that automatically redirect the user to the malware download.

`IFrame` - `IFrame` is a generic name for malicious `IFrame` tags embedded in HTML pages, which may redirect the browser to a specific URL location with malicious software.

`Bundpil` - `Bundpil` family variants are worms that usually spread via removable devices. They tipically contain an URL address, and they try to download several files from that address, executing the downloaded files afterwards. These worms usually delete some important files from the computer of the victims.

3.3.3 Top 10 Families by Kaspersky Lab for the 3rd Trimester 2013

Table 3.3 presents some of the top families that have been detected by Kaspersky Lab on the 3rd trimester of 2013 on end-devices [CG13]. The families presented in the previous table are briefly

Table 3.3: Top 10 families detected by Kaspersky Lab on the 3rd trimester of 2013.

Order	Family
1	Win32/AutoRun
2	Win32/DelBar
3	Win32/Sality
4	Win32/Bromngr
5	Win32/CVE-2010-2568
6	Win32/Bromngr
7	Win32/Agent
8	Win32/Debris
9	Win32/Starter
10	Win32/WebCake

described below, however, as before, the families that where previously subject of discussion are not included below:

`DelBar` - Samples of the `DelBar` family run secretly in the system, pop up ads, change search results, and collect private information of the user.

`Bromngr` - The `Bromngr` malware family consists of a series of malware variants that are usually designed to make money, by generating network traffic or displaying potentially unwanted advertisements. This kind of malware is usually able to perform browser hijacking, install toolbars or simply display pop-ups on the computer.

`CVE-2010-2568` - This exploit is catalogued as a Microsoft Windows OS vulnerability on `.lnk` and `.pif` shortcut files. This vulnerability allows for the local users or remote attackers to execute arbitrary code via these type of shortcuts, given the fact that they do not properly handle the icon displaying on Windows explorer.

`Bromngr` - This family appears two times on this list because two different variants made it to different positions of the rank. The behaviour is nonetheless the same.

`Agent` - Samples of this family work like the ones from the `DelBar` family, popping up unwanted ads on the computer of the victims.

`Debris` - The `Debris` family of malware represents a critical infection and a major threat to computers with the Windows OS. Samples of this family not only remove important system files, but also restrict the user access to the most visited data folders. Besides that, they can also block important system utilities such as the task manager and registry editor.

`Starter` - Members of this family, create an unauthorized user account on computer systems, and add administrator privileges to that account, defining it as a *Remote Service Account*.

`WebCake` - Samples of this adware family trigger pop-ups with ads while browsing the Internet. They can be downloaded from their official websites, or installed via a third-party software.

3.3.4 Top 10 Families by ESET in Portugal on January 2014

Table 3.4 presents some of the top families that where detected by ESET in Portugal on January 2014 on end-devices [ESE14]. The malware families included in the aforementioned table are

Table 3.4: Top 10 families detected by ESET in Portugal on January 2014.

Order	Family
1	HTML/ScrInject
2	HTML/Iframe
3	Win32/Waski
4	Win32/Boaxxe
5	JS/Yontoo
6	JS/Agent
7	Win32/Fareit
8	Win32/MultiPlug
9	Win32/PCMega
10	Win32/VMProtect

described below, except for the `ScrInject`, `IFrame` and `Agent` families that where already explained before:

`Waski` - Malware within this family is able to silently install other programs without consent. This can include the installation of additional malware or components to an infected computer, or even legitimate software.

`Boaxxe` - `Boaxxe` is a family of trojans that install themselves as Browser Helper Objects (BHOs). They are able to contact remote websites to download and run arbitrary files.

`Yontoo` - `Yontoo` describes a family associated with adware behaviours. As such, its members show ads while the user browses the Internet. This malware can be downloaded from the official application website, or via third party applications.

`Fareit` - This family has many variants, ranging from password stealing malware to trojans that can be used to perform DDoS attacks on remote web servers.

`MultiPlug` - This family is very similar in purpose and functioning to the `Yontoo` family.

`PCMega` - Another adware family, whose members pop up ads on the computers of the victims without their consent.

`VMProtect` - Variants of this family usually disguise as fake applications that look just like the usual anti-spyware tools. However, these variants infect the system without the consent of the user, performing several actions to damage system programs and resources. They are typically able to modify system registry settings. Then they show pop-up windows to the user, claiming that some programs are infected and that, in order to remove them, the user should buy the paid version of the sofware. This family is therefore seen as ransomware.

3.4 Malware Families and Variants Tree

The contents of this section reflect an attempt to give an overview of the most known and detected malware families, and their respective variants, using a graphical representation in the form of a tree. This representation is a snapshot of the malware state on February, 2014. This is a very dynamic area and, as such, it may become outdated quickly. To keep it up to date would require a constant effort. At the time of the writing of this dissertation, this tree may actually not be 100% accurate. The genealogical malware family tree, included in figure 3.1, was mostly built resorting to the Virusview [Lab13a, Lab13c, Lab13d, Lab13b] and Microsoft security portal [Mic14] websites. Notice that the figure is included in an A3 page format with landscape orientation, for the sake of readability.

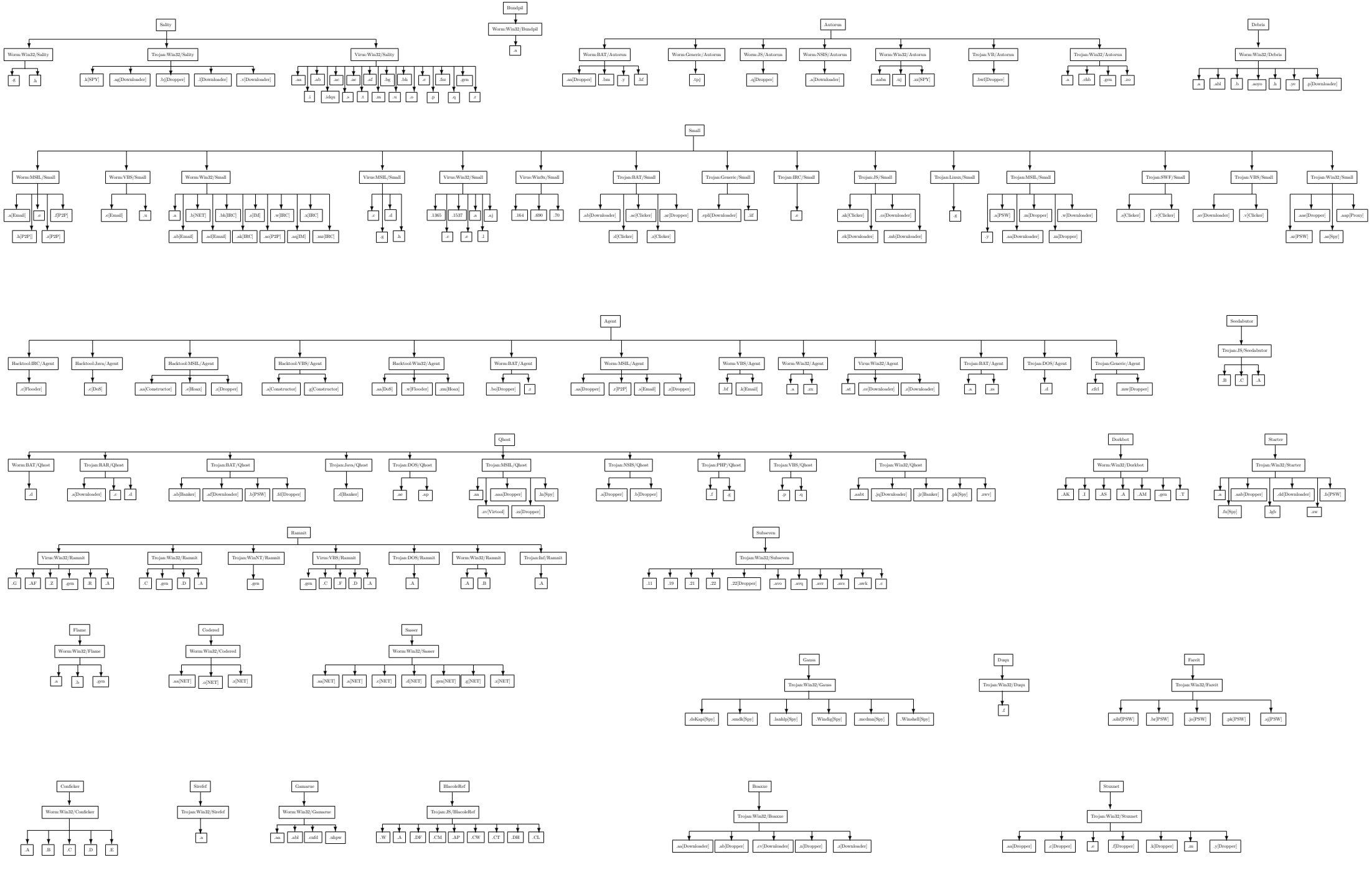


Figure 3.1: Malware Family Variants Tree.

3.5 Malware Behaviours, Hiding and Data Encoding

In order to consistently document malware, it is important to take a look at some of its most impressive features and turns them into unique and really sophisticated pieces of engineering. In this section, a brief revision of the most important and sophisticated features of these programs is going to be performed. Some malware behaviours have already been described in Chapter 2. Nonetheless, it is important to focus on a few more aspects of those behaviours.

3.5.1 Malware Behaviours

Two of the most common types of malware in the wild are `Downloaders` and `Launchers`. The `Downloader` behaviour translates into the ability of downloading another malware piece from the Internet and find a way of executing it on the system. Usually, `Downloaders` use the Windows API `URLDownloadtoFileA` to obtain the malware file, and then they usually call `WinExec` to execute the new malware piece. `Launchers`, also known as (a.k.a.), `Loaders` correspond to threats present in any executable that install malware for immediate or for future covert execution. Most of the times, `Launchers` contain the malware that they are programmed to run.

Malware with the `Backdoor` behaviour provides an attacker the ability to remotely access the computer of the victim. These are very common types of malware, and they possess a wide range of functionalities. Usually, if a `Backdoor` is present in the source code, the malware sample is not going to need to download additional malware or code. `Backdoors` usually communicate over port 80 using HTTP. Under normal network conditions, malware uses the HTTP over port 80 as the main communication channel, since it is easier to merge with normal traffic and pass undetected this way. `Backdoors` are usually able to perform a series of activities, namely:

1. manipulate registry keys,
2. enumerate display windows,
3. create directories,
4. search files;
5. etc.

The analyst can then determine which of the next three activities is exhibited by a `Backdoor`, by taking a look at the functions that are used and imported:

- The *Reverse Shell* is the name given to the connection that is established with the infected machine, which provides shell access to the attackers. Reverse shells might be found as standalone malware or as a component of a more sophisticated `Backdoor`. Reverse shells

allow the attackers to perform all sorts of commands as if they where on the infected computer. Windows reverse shells are implemented usually resorting to `cmd.exe` and they are commonly classified as basic and multi-threaded. The basic Windows reverse shells are the most popular ones because they are the simplest of both of them. The multi-threaded version requires the creation of a socket, two pipes and two threads. In order to create pipes and threads, there should be some calls to the `CreateThread` and `CreatePipe` functions. The need of threads and pipes is a common strategy amongst malware authors to manipulate or encode the data travelling through a socket. After the `CreateProcess` is executed, the malware will launch two threads. One of them will be responsible for reading from the standard input pipe and writing to the socket, while the other one will read the socket data and write it to the standard output pipe.

- Remote Administration Tools (RATs) are usually used to manage a computer or a set of computers remotely. These tools are only used in targeted attacks with specific objectives in mind, like stealing a particular information or gaining access to other resources present on the network of the victim. Usually, communications are performed over port 80 or 443, which are associated to HTTP and Hypertext Transfer Protocol Secure (HTTPS), respectively, because they are two of the most common ports on the network traffic. This technique is based on the client-server architecture. The client side sits on the computer of the attacker and acts as Command-and-Control unit. The server is running on the computer of the victim.

- *Botnets* are usually a set of compromised computers, also known as zombies, which can be controlled by a remote entity, typically by resorting to a server called botnet controller. The main goals of botnets are to increase, spread other malware, spam or perform DDoS attacks. Nowadays, botnets are the main cause of failure of important websites.

It should be noted that a botnet is not a RAT, since botnets have been known to infect and control millions of hosts simultaneously, while RATs are used to control fewer hosts. On a botnet, the attacker can command all the infected computers simultaneously, while RATs, are targeted to one victim at a time. RATs are used in targeted attacks while botnets are used in mass attacks.

`Credential Stealers` are created with the intention of stealing some information and, on an era in which every device might be online, the access credentials are the most desired information. Usually credential stealing is implemented on three types of malware:

1. programs that steal the credentials after system login;

2. programs that dump the system memory to search for password hashes in order to use them directly or to crack them offline; and

3. programs that keep track of the keystrokes.

One common way of stealing credentials after system login is the Graphical Identification and Authentication (GINA) interception on Windows XP. This Microsoft functionality was built with the sole purpose of allowing legitimate third parties to customize the logon process. That customization could be for providing support for smart cards or Radio-Frequency Identification (RFID). Usually, malware authors can take advantage of this OS functionality, which is implemented via a Dynamic Link Library (DLL) called `msgina.dll` and loaded by the `Winlogon.exe` on every login process. The malware author can place a malicious DLL under the registry key, which configures which DLLs should be loaded by the `Winlogon.exe`. By setting a malicious DLL in the middle of the loading order, the communications from the `Winlogon.exe` to the `msgina.dll` will be intercepted and stored on disk or sent to the malware author.

Hash dumping is the most widespread name for the second form of implementation of credential stealers, and consists on performing a dump of the OS memory in an attempt to grab password hashes so that the malicious user can crack them offline, or use them on an attack often named as Pass-the-Hash (PSH) attack. Hash dumps can be obtained resorting to *Pwdump* and PSH Toolkits. Malware exhibiting this behaviour tipically uses a call to the `lsass.exe` process, which has the necessary privilege level, as well as access to many useful API functions.

The last better known option for credential stealing with malware is the implementation of a *keylogger*. These programs can have different variations on the way of capturing the keystrokes, but they can be typically categorized into three main classes. One of the best and most sophisticated ones is called kernel-base keylogging, and is characterized by usually being inside a rootkit, and act as a keyboard driver. The other classes are very similar and they are both called the same: user-space keylogging. Nonetheless, the technique in which they rely on is not the same. One can be performed with *hooking*, which uses the `SetWindowsHookEx` API to notify the malware each time a key is pressed, and the other with *polling*, which relies on two APIs to work correctly: the `GetAsyncKeyState` and the `GetForegroundWindow`. By searching for the imported libraries, or for typical strings, it is possible to detect this kind of malware.

After gaining access to the system of the victim, malware often tries to keep itself on the system as long as it can. In order to accomplish that objective, it often contains the so called `persistence mechanisms`, including:

- One of the most common persistence mechanisms uses the *windows registry*. This Windows structure might be used by malware to store some configurations and gather information. By setting a key on any Windows registry autorun location, malware is able to start up every time the computer is rebooted. The most important autorun locations are the `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run`, which is responsible for run-

ning all the applications that should start with Microsoft Windows, *AppInit_DLL* location (`HKLM\SOFTWARE\Microsoft\Windows NT\CurentVersion\Windows`), which is responsible for loading all the DLLs that are set on this key, to every process that requires `User32.dll`, *Winlogon notify location* (`HKLM\SOFTWARE\Microsoft\Microsoft\Windows NT\CurrentVersion\Winlogon`), which can save data regarding the system time at which the application should be started and, at last, the *SvcHost DLLs location* (`HKLM\System\CurrentControlSet\Services\ServiceName`), which contains the services that should run by the `SvcHost` application.

- Another way to gain persistence on the system is by performing changes to an existing popular binary. These binaries are then called *trojanized system binaries*. The changes are made in order to force the system into executing the malware the next time that the binary is executed or loaded. A binary is usually modified by changing its flow of execution, i.e., the attacker sets a jump to another location within the byte code.

- On the other hand, the *DLL load-order hijacking* does not require any Windows registry entry and no particular change to any system binary. This mechanism is therefore stealthier. The trick is to know the Windows default default order for loading DLLs. On the Windows XP OS, this is an easy objective to achieve. If there is a software that runs from the `Windows` directory and requires a `Windows\System32` DLL, the default order is followed, and therefore, the `Windows` directory is going to be searched for that DLL. However, if the malware writer has knowledge about this way of functioning it is possible to create a fake DLL with the name of an existing one and put it into the `Windows` folder. This way, the legitimate program will run with the fake DLL and continue its execution. Often, this technique involves calling the true DLL from the malicious one in order to allow the system to flow correctly.

- When a user does not provide administrator privileges to the malware, it will need to perform a *privilege escalation* in order to gain full access to the system and deliver its full payload. One way of gaining privileged access is by setting the access rights of the owner to enable `SeDebugPrivilege`. This privilege was created in order to provide system-level debugging and, by default, is given only to local administrator accounts. To give this privilege to any user is the same of granting that user with local administrator privileges. To detect changes to this particular privilege, the analysts should search for a call to `AdjustTokenPrivileges`.

3.5.2 Malware Hiding Mechanisms

Users evolved along with the computers. For example, almost every user knows how to list the Windows processes and, as such, malware authors had to try to find ways into deceiving them. This subsection is therefore intended to present an overview about the mechanisms used to hide malware on the system.

`Rootkits` comprise one of the most popular means to hide malware activities. They can be implemented in many ways. However, most of them, work the same. They usually modify the functionalities of the OS kernel. These changes have the objective to keep files, processes, network connections and other computer resources invisible to other programs. Two common `rootkit` techniques are the *Import Address Table (IAT) hooking* and *inline hooking*.

`Launchers`, are executable files that install malware for immediate of future covert execution. Sometimes, they already contain the malware that is to be executed on the resource section of the PE file and, once the launcher is executed, it extracts a PE file or a DLL before launching it. Usually, calls to functions like `FindResource`, `LoadResource` and `SizeofResource` are used in this process. Launchers usually perform privilege escalation, so they usually have pieces of privilege escalation code also.

The most popular way of malware concealment is `process injection`. As the name implies, `process injection` involves the injection of malicious code into a legitimate process, leaving it with the burden of delivering the malicious payload. These techniques can also be used to try to bypass host-based firewalls and other specific security mechanisms that rely on the processes of the system. Most of the times, they involve calls to `VirtualAllocEx` and `WriteProcessMemory`. Two common process injection techniques are as follows:

- *DLL injection*, is the most widespread form of malware concealment. *DLL injection* consists in injecting code into a remote process that calls `LoadLibrary`, forcing a DLL to be loaded. Once loaded, there is a call to `DllMain` in order to find its main function, which makes the malicious function to be executed within the legitimate process, with the same privilege level. Every action that is performed by this DLL will appear as if it was originated from the legitimate process.

- *Direct injection* is basically the same thing as *DLL injection*, however, it does not involve injecting a DLL into a running process. Instead, it injects code directly into the selected process, thus not requiring to load a DLL into that process. As such, it is a more flexible technique.

`Process replacement` involves replacing a process in memory with another one. This is normally achieved by substituting the memory space in which a process is operating, with malicious code, thus forcing a legitimate process into processing a malicious one. This is a sophisticated technique, but if well done, it can overcome the problems of crashing a process, which are common with `process injection`.

The windows message-handling mechanism might have points where an application can install a subroutine, in order to monitor the traffic in the system, and perform a few processing tasks,

before certain types of messages reach their target. Inserting monitors at these points is called `Hook Injection`. Hooks can be classified as local, which are used to observe or manipulate messages destined to an internal process, and remote, which are used to observe or manipulate messages destined towards remote processes. `Hook Injection` corresponds therefore in the insertion of a malicious DLL on the message traffic flow of the system. This technique might serve two purposes for malware authors. It can be used to run a particular malicious function every time a certain message is intercepted or even to make sure a particular DLL is loaded on the memory space of the victim. Hooks are also a good way of implementing a *keylogger*, and they are usually found under these malware techniques. In order to identify hook injection on software, the analyst should search for `SetWindowsHookEx`, which is the procedure used to perform remote windows hooking. On top of this, the hook procedure should always have a call to `CallNextHookEx`, in order to ensure that the next hook on the chain is called and the system keeps flowing properly.

`Detours` is a library developed by Microsoft with the objective of allowing the implementation of extended functionalities on Microsoft applications and OSs. However, malware developers started using it in order to perform changes to the import table on the PE files, attach DLLs to existing programs, and add function hooks into running processes. Even despite this library allows the malware authors to perform any of the above three activities, usually they use it as a means to insert new DLLs into an existing binary. This library works by creating a new section on the PE file header called `.detour`. This section usually has the original PE file header with a new IAT. The malware author then uses the detour library (with a call to `Setdll`), or any similar one, to add new entries to the import table.

3.5.3 Malware Data Encoding

This section aims at introducing the most common, and known, forms of content modification with the purpose of hiding malware. Most of the times, malware uses encoding techniques in order to fulfil multiple purposes. One of the main objectives is to hide the network communications, but they are also used to hide configuration information, save information to a temporary file before obtaining it, store strings and to disguise malware as a legitimate software. The malware analyst should first try to find the encoding functions to then try to decode the data. Subsequently, a few data encoding techniques are briefly presented.

Many of the simple encoding techniques are old but they still find use in nowadays, mostly because their computational cost is lower than complex techniques. They are commonly used in malware nowadays, mostly due to the fact that they are small enough to be used in environments where the code length is limited, like shellcode. Sometimes, `simple ciphers` can be less obvious than the more complex alternatives, and because they are smaller and simpler, they present a lower overhead, which translates in a smaller impact on performance. Two of the most common

simple cipher encoding methods are based on the Caesar Cipher and on the Vigenère cipher over bits (Exclusive OR (XOR)).

`Base64` is the most widespread encoding technique used by malware. It is used in order to represent binary data in an American Standard Code for Information Interchange (ASCII) string format. It was originally developed in order to encode e-mail attachments for network transmission. However, it is now used for most of the HTTP and eXtensible Markup Language (XML) network traffic. `Base64` converts binary data to a limited set of 64 characters, effectively resulting in a bigger message: every 3 bytes of binary data are encoded into 4 bytes of encoded data. The process of encoding the raw data into `Base64` is standardized. It uses 24 bits (3 byte) blocks. The first step is then to find the 8 bit binary code of the first three letters. Once it is found, the 24 bits that represent the three letters, are partitioned into 6 bit codes. Each of those codes will be represented by a letter on the `Base64` code. The process of decoding the data is similar. Each `Base64` character is transformed into a 6 bit code, and all of the 6 bits are sequentially placed. The bits are then read 8 bits at a time, converting them to ASCII.

Cryptography was one of the areas which evolved more in the last few years. New means of encoding data, which leverage crypto related primitives and the current computer processing power. This led to the development of encoding algorithms for which the decoding process is practically impossible without the knowledge of a key. All the `simple ciphers` mentioned earlier are vulnerable to brute-force attacks. However, modern cryptographic algorithms overcome this problem. Nonetheless, malware does not always take advantage of these mechanisms, specially due to the following facts:

1. These algorithms are larger than simple ciphers, in terms of code footprint and, as such, malware may need to have a link to existing code on the host, which reduces portability;

2. The usual cryptographic libraries are easily detected, either by the imports, or by the identification of cryptographic constants;

3. When using a symmetric encryption algorithm, hiding the key poses an additional concern.

Another interesting aspect of these algorithms concerns the size of the encryption key. These keys should be as large as possible, but malware needs to be as small as possible. In order to find cryptographic encoding, the malware analyst should focus mainly on strings and imports at first. For instance, if the malware resorts to the `OpenSSL` library, in order to encode its network communications, the string *OpenSSL* must be visible. However, if the string is not there, there is still the possibility of looking at the malware imports, and search for the ones related with cryptographic functions. Most of those functions in the Windows OSs begin with the prefix `Crypt`, `Cryptographic Provider (CP)` or `Cert`. Other potential approaches rely on

searching for cryptographic constants using specific software like the *FindCrypt2* and the *Krypto ANALyzer*. Another approach consists in measuring the entropy value of the malware files. Data encoded using modern encryption techniques will have a very high entropy value.

Interestingly, malware authors tend to use a combination of encoding mechanisms to demotivate malware analysts. This combination uses normally simple encoding schemes with small size and no apparent use of encryption. They usually combine `simple ciphers` and `cryptographic algorithms`, e.g., they can encode the malware using the XOR encryption, and then the `Base64` encoding, leaving the analyst with a harder analysis case on his hands. One of the most used techniques to obtain the decrypted data is to allow the program to perform its own decryption while it executes. This process is called self-decoding and relies on the debugging of malware, recurring to a debugger.

3.6 Anti-Disassembly and Virtual Machine Techniques

On this section, an overview over the techniques that malware may contain in order to difficult the job of a malware analyst is performed. The techniques herein introduced are specifically designed to prevent analysts from disassembling malware or running the samples on virtual machine environments.

3.6.1 Anti-Disassembly Techniques

Anti-disassembly techniques are used to cause disassembly analysis tools to fail. These techniques usually rely on the knowledge and expertise of the malware developer when crafting the malicious software. Developers can insert these modules using a separate tool during the build process, or insert them into the source code of the malware. Code armed with these features, forces the malware analys to have more knowledge on this subject, and can turn the analysis into a large research and reverse-engineering project. On top of that, malware armed with anti-disassembly features can also be hard to detect and classify by automatic analysis processes, like heuristic mechanisms of AVs. Some techniques of anti-disassembly are introduced below.

Code disassembly is not simple and consists in representing sequences of executable code in disassembly representations, namely in the assembly language. However, the same sequences of executable code may result in different representations, and some of them may be invalid. The invalid representations are inserted to obscure the real functionality of the program. Anti-disassembly techniques take advantage of this particular flaw. Malware developers usually try to create a sequence that tricks the disassembler in order for it to show the wrong representations of the executable code. In order to understand this topic better, it is important to know the two different kinds of disassemblers, namely:

The *Linear disassemblers* work by iterating through each block of code, disassembling one in-

struction at a time. This is a widely employed strategy by debuggers. Linear disassemblers do not use flow control in order to disassemble the code. Instead, they use the size of the previously disassembled instruction to obtain the byte that they should disassemble next. This approach comes with a few drawbacks, like probably disassembling more code than needed, because it will keep disassembling until the end of the buffer, even if the flow of the program does not go there. The code of a PE file is usually under the `.text` section. However, not everything in there is code (for example, pointers are also found in that section). Malware developers exploit this fact and sometimes insert pointers in such a way that, when a linear disassembler gets there, and connects those pointers to the main code, produce bad results.

Flow-oriented disassemblers are the most popular disassemblers nowadays, with IDAPro 6.3 being an example of such software. The main difference between them and linear disassemblers is that flow-oriented disassemblers do not iterate over the entire buffer. Instead, they examine each one of the instructions on the buffer, disassembling only the set of instructions that are called and executed during execution. These disassemblers make choices and assumptions, which may seem unnecessary. However, this operation mode can make the difference when anti-disassembly mechanisms exist in the code.

Now that the disassembler types are explained, a few anti-disassembly techniques can be discussed.

One of the most used techniques to fool disassemblers consists in setting an unconditional jump (often referred to as *jump instructions with the same target*). An unconditional jump is a jump which will go to the same place, independently of entering the true or the false branch of an if clause. This way, malware developers can hide or obfuscate some parts of the code. For example, by placing a call instruction before the location pointed out by two branches of an if clause can cause the linear disassemblers to understand that instruction as legitimate when it is never used. Additionally, if the injected instruction has parameters, the next instructions may be understood as values, preventing the disassembler from seeing the true instructions.

Another commonly employed technique is named *jump instructions with a constant condition*. It consists on inserting a false branch into the code, in the sense that the condition that guards that branch is always false at the point of execution. Since most of the disassemblers will look into the first branch, they will trust that code more. Sometimes that false code is conflicting with the code contained in the true branch, thus making the process fail.

3.6.2 Anti-Debugging Techniques

Sometimes, disassembling does not give the analyst enough information. In such case, executing the malware sample on a controled environment may help to obtain more useful information.

The analyst may then try to debug the malware. However, malware developers employ a series of different techniques in an attempt to pledge the analysis resorting to this particular approach. Some of the most important and known techniques that allow malware developers to compromise debugging are exposed below.

One of the easiest and obvious ways of detecting debuggers is to use the `Windows API`. It has several functions that can be used to check if a program is being debugged. The easiest way for malware analysts to overcome anti-debugging function calls is to manually change malware during execution in such a way that these functions are not called and the proper flow is followed. However, there are other approaches that might be taken, e.g., hooking the API anti-debugging methods as with a rootkit. Some methods of the Windows API to perform anti-debugging techniques are `IsDebuggerPresent`, `CheckRemoteDebuggerPresent`, `NtQueryInformationProcess` and `OutputDebugString`

Because using the windows API is the most obvious and simple method, malware developers nowadays avoid using this option, specially because the calls performed can be hooked by a rootkit and may return false information. Due to this, some malware developers prefer to use a different approach when implementing anti-debugging techniques. `Manually checking structures`, even being a more complex procedure, is the most common technique nowadays. It consists on performing a search on the several flags which are set on a structure storing all the processes that are being currently executed by the OS. That structure is called Process Environment Block (PEB). The flags of this structure, which can usually be consulted, in order to find if the malware is currently a target of any debugging software are `BeingDebugged`, `ProcessHeap` and `NTGlobalFlag`.

`System residues` is related the information left on the system after an application exits. These residues can include windows registry keys, files, directories and residues in the current memory. However, the most common check performed by malware is by resorting to the `FindWindow` API method. All of these checks are performed so as to find common debugger resources and strings.

3.6.3 Anti-Virtual Machine Techniques

This subsection introduces some of the techniques that malware uses to detect execution on a virtual machine and some of the most common behaviours it may take in the case it finds itself on a virtual environment. These techniques are employed mostly on bots, scareware and spyware, specially because honeypots often use virtual machines. Anti-virtual machine techniques are not so important nowadays, specially due to the fact that virtualization is being increasingly more considered as a standard by the industry. Therefore, malware developers now know that, just because a machine is a virtual machine, it does not necessarily mean that it is not a considerable victim. Most of these techniques aim to the VMware software. The information presented next

refers directly to anti-virtual machine techniques programmed against this software:

VMware string - Like any program installed on Microsoft Windows OSs, VMware leaves some residues on the computer, which can be used to detect the software. This takes us to the first technique to detect a virtual environment. By performing a search on the process listing for the string *VMware*, it is possible to know that a related application is running, and thus, a virtual environment is being executed.

Registry Key - Another feature which can be used, is the registry key for the `VMwareService.exe`.

Instalation Directory - Other way is to search for the VMware installation directory.

Network Interface Card (NIC) virtualization - The NIC virtualization that must be performed, in order to provide Internet access to the virtual machine, is considered an advanced technique. VMware must provide this NIC with a Media Access Control (MAC) address. Depending on the settings of this virtualized component, it might be possible to identify VMware. MAC addresses are specific to each manufacturer and, in this case the VMware NICs MAC addresses, change from version to version. For example, a NIC with a MAC address starting with `00:0C:29` is usually associated with a virtualized NIC within a VMware installation.

Motherboard - Malware can also try to detect VMware by the motherboard. Most of the times, all of these checks can be easily surpassed by patching the malware binary, i.e., by using an hex editor to change the strings used in the searches. This way, the comparisons will all be false, and the malware will think that it is not being executed in a virtual machine.

3.7 Conclusion

A brief study over the malware naming convention was reported in this chapter as an introductory note to the presentation of the most detected malware families by the top brands of the malware fighting industry. Afterwards, a few malware behaviours were described with more detail. The family tree for the most popular samples was then included, fulfilling one of the objectives of this work. The discussion evolved to the explanation of the mechanisms used to guarantee persistence and obfuscation. This included the discussion of malware hiding and encoding mechanisms. It was important to study these subjects because they provide good hints to the malware analysis process, presented afterwards in this Ms.C. dissertation. They also constitute topics of interest for analysts working on this area. Given the techniques described herein, it seems now important to understand which tools and techniques can be used to protect computer systems against malware.

Chapter 4

Fighting Malware

4.1 Introduction

Many people in developed countries rely on computers for leisure purposes, to work, shop and communicate. Almost every user nowadays, has some sensitive information on its computer, and most of them use computer devices to access more sensitive information daily. So, malware poses a real and very critical threat nowadays, and it is of utmost importance for the user to protect infrastructures and data. Because of this requirements, there are a lot of companies centered on this subject, which try to deliver security solutions both to companies and end-users. The amount of solutions to detect, analyse and remove malware has been growing and improving with the years also. In this chapter some software and techniques to detect, analyse and remove these programs are introduced and explained.

This chapter is structured as follows. Section 4.2 is focused on detection and removing tools, while Section 4.3 describes dissection tools.

4.2 Detection and Removing Tools

There are several tools and techniques to detect and remove malware. Nonetheless, before describing some of them it is important to discuss some practices that humans can take to avoid being infected with such malicious software. The next section discusses precisely this subject.

4.2.1 Knowledge and Security Concerns

There are several behaviours that a user can adopt to avoid being surprised by malware [Wik14a]. Sometimes, even the most experienced users get surprised by trojans or worms on their computers, since this threats do not depend as much from the human behaviour. Nonetheless, threats that depend on user actions to disseminate can be fought with proactive measures. Those behaviours and actions can be summarily described as follows:

- The user should never click on a link or attachment in an e-mail if he or she is not absolutely sure that the source is trustful. In case of doubt, the user may send an e-mail to verify if it is legitimate before performing any action associated with it.

- If the user uses an e-mail client like outlook, it is recommended that the *Allow image previews* option is disabled because, often these programs automatically load attachments

for convenience, thus making it impossible to decide whether an e-mail attachment is safe or not before it gets opened.

- The user should be careful when opening attachments of untrusted e-mails, specially because of the double extensions problem. Sometimes, hackers, hide the malicious code under the names and extensions of other types of files, like for example, `.txt.vb` or `.jpg.exe`. Some OSs hide the file extensions, making it more difficult to detect such situations.

- The user should avoid using removable drives from other users. For example, connecting the USB drive of someone else into the computer can trigger an infection.

- Users should avoid Internet pop-ups. Sometimes, these pop-ups are attractive but lead to malicious sites containing malware. AV related propaganda should be always avoided. In case of being addressed by such pop-ups, it may be a good idea to clean the browser cache and temporary files.

- The user should not trust e-mails requiring for personal information, namely passwords or bank-account numbers, even if they come from an apparently trusted sender. The transmission of sensitive information should be done in person (e.g., in the bank premises) or using a trusted secure communication means (e.g., using a verified Transport Layer Security (TLS) connection).

- The user should avoid using file-sharing applications like eMule and BitTorrent, because they are often used by attackers to share malicious contents with the name of interesting movies or music.

- The user should be aware of phishing schemes also. Most phishing schemes use a description of a situation that puts the user as the main heir of a big fortune, or as the winner of a contest in which he may have not participated, etc. The second most common form of phishing relies on the impersonation of a reliable entity to ask for personal data of the user, which falls into the situation described above.

- If the user is asked to install a certain software when visiting a website, and the user is unsure of what exactly the software is needed to, then the instalation should be aborted and the program should be searched in the Internet first.

- The user should pay particular attention to the URLs of websites. Many times, phishing schemes use similar webpages and similar URLs to trick the user into the insertion of his own personal information on the *fake* websites. Sometimes the URLs are similar also.

- Usually, if a web-application manipulates user data or monetary information, it is certified by a company that produces digital certificates (e.g., VeriSign). The certificate should be checked in this case.

- Remote assistance should only be provided to reliable persons. Some malware can be used to infect the remotely assisted computer if the assisting computer is infected as well.

- The user should consider using different Internet browsers, after researching the Internet for this security topic. Some browsers are more secure than the ones provided natively with the OSs.

- Most system updates fix security problems and block spyware and viruses. As such, the system should be updated regularly.

- Firewalls should be installed and activated on end-devices and network infrastructures.

- The user should also have an up-to-date AV and anti-spyware program installed. The user should also consider performing a full system scan often.

- When installing freeware, shareware or similar applications, never allow the installation of additional unknown programs or tools that may come in the package, because they may be malicious.

The user should always keep a backup copy of important files. If a backup does not exist, the user risks losing everything when a virus or spyware hits the system.

4.2.2 Firewalls, IDSs and IPSs

As mentioned before, there are several good practices a user can use to avoid being caught by malware. Nonetheless, it is still possible for a threat to spread based on the system vulnerabilities. It is thus important to discuss the software or hardware systems that can be used to detect, prevent or contain the spreading of malware, and minimize their effects. Firewalls, Intrusion Detection Systems (IDSs) and Intrusion Protection Systems (IPSs) comprise three of these systems.

4.2.2.1 Firewall

A Firewall is a system that forms a barrier between the private network and the outside world. These systems are often installed near network routers or these devices are part of the firewalls themselves. However, it is possible to find solutions comprised by separate hardware and separate software. The private network may be the computer of the user (if the firewall is a software

system installed on it) or an entire network. The aim of these systems is to prevent *outsiders* to enter the firewall protected perimeter, preventing them from gaining unauthorized access to the protected systems. These systems work by monitoring incoming and outgoing network traffic and by deciding whether to forward or block the communications depending on a predefined set of rules. Those rules may vary according only to the protocol of one of the Open Systems Interconnection (OSI) layers, to the source and destination ports, to the source and destination IP addresses or to the direction of the communication. Typically, firewalls work at the session layer. Firewalls usually do not return the dropped packets to their destinations, because this would result on an increase of bandwidth use. They typically have a predefined setting, which is to deny all the network traffic they process. So, if the system is badly configured, the network may not function properly and false positives may be raised during the monitoring process.

With the growth of malware and suspicious communications, today, most enterprise firewall solutions use stateful firewalls, which monitor the state of network connections over a period of time, so that the system administrator can create the rules of legitimate data packets for each connection and, this way, being able to only allow known connections and deny all others. Basically, these firewalls allow incoming packets that are specific responses to requests from internal hosts. Any packet that is not a specific response, is blocked unless it is permitted by another rule. Network Address Translation (NAT) is typically provided by firewalls also. NAT is a mechanism for translating private IP addresses into public IP addresses. This feature allows the protected network IP addresses to be hidden to outside users, making it hard for outsiders to communicate directly with the internal network hosts.

On top of these firewall models, with the proliferation of attacks exploring application vulnerabilities and special features, some new models that where specially thought to try to fight these kind of attacks appeared. Nowadays there are firewalls acting upon protocols of the OSI application layer and others based on circuit filtering. Applicational firewalls are able to filter the network traffic not only by analysing the transport layer information, but also the application layer data, and pose as an intermediate node in the communications. In this case, the firewall receives commands and messages as if it was the end-node, filtering out the ones that are not permitted. As such, these firewalls are usually called proxy servers. If a host inside the security perimeter wishes to communicate with an Internet server, the communication is established with the applicational firewall, and the communication with the Internet server is then established by the firewall. Even though these firewalls provide more security than the packet filtering ones, they are also more expensive in terms of computational resources and, as such, financially too.

A circuit filtering firewall works like an applicational firewall, but they only intermediate Transmission Control Protocol (TCP) sessions. When an internal host wants to connect to an external server, the connection is performed to the firewall, which will then establish another TCP con-

nection with the server. These firewalls do not look into the contents of the packets and, as such, they are not as expensive as the previously mentioned ones. By operating at this lever, they can see if the session establishment is legitimate, and open a pipe proxy to the desired destination in the successful case. After opening the pipe, the communications are no longer analysed during the livecycle of the TCP connection. While these firewalls prevent the establishment of connections to untrusted sources (if they are blacklisted) and also avoid that an attacker communicates directly with the internal host, it does not prevent a malicious payload from passing on already established connections [Cir08, Pas11, CPST14, Dow07, Wik14b, Ina14].

4.2.2.2 IDSs

It is common to compare firewalls with IDSs. A firewall does not typically inspect the payload of the packets in search of malicious contents. They normally block traffic or connections based on rules. On the other hand, IDSs can look inside the network packets, files and attachments and run detection algorithms on that data. The maintenance of the most complex IDSs is done by specialized personnel. The main objectives of these systems, is to detect, launch an alert and sometimes perform some countermeasures towards computer security threats. The quality of these systems is typically measured resorting to the number of false and true positives, false and true negatives. Too many false positives hinder the trustfulness of the system, while too many false negatives make them useless. The architecture of an IDS is usually defined using 4 modules:

- At the bottom layer of IDSs are the so called *Event (E) boxes*, which are responsible for capturing events. These events may come from several sources, namely network traffic, system calls or logs, depending on the purposes and on where the system is installed. It is possible to configure separated sensors on different parts of the network which are able to communicate with the main IDS, either sporadically or by demand. Other interesting events concern information regarding users with initiated session, processes in execution, processing charge, file modifications, etc.

- Every information captured on the sensors is then submitted to the *Analysis (A) boxes* module, which are responsible for analysing the data, in order to identify intrusions and their designations. Signature, behaviour and heuristic based algorithms are included and executed in this boxes.

- The events that are considered interesting by any of the A boxes algorithms are then stored, for future analysis, by the mechanisms implemented in the *Storage (D) box* module, along with the result of the analysis provided by the A box module.

- The IDS may include mechanisms to respond to detected threats. If so, they are implemented in the so called *Countermeasure (C) or Reaction (R) boxes*, which receive the outputs of the analysis of the A boxes and act accordingly. These actions may include issuing alerts towards the administrator, write on the system log or reconfigure other defence systems like firewalls (e.g., specific rules may be set on the perimeter firewall to prevent traffic from a given source to enter). These measures can also include the isolation of attacked systems and counter-attacks, though the later are typically not advised.

IDSs are typically classified according to several operational aspects. Their basic architecture is nonetheless shared by all classes. The operational aspects for which IDSs are classified are: (i) the Detection Method, (ii) the Events Source, (iii) the Timing, (iv) the Reactivity and (v) the Distributivity.

According to the detection method, an IDS can be classified as *signature based* or *behaviour based*. A signature based IDS searches for known attack patterns (signatures) on any of the aforementioned events. These signatures may be strings, certain known patterns on data transmission, etc. The major drawbacks of these systems concern the constantly growing size of the malware signatures database and the computational costs. Since they rely on a signature database to detect intrusions, they have the disadvantage of only detecting known intrusions also. Nonetheless, they are very accurate and capable of pinpointing the name of the detected menace. On the other hand, behaviour based IDSs are not as computationally expensive and can be used to detect unknown attacks, because they elaborate on a different philosophy. The idea is to model the normal behaviour of the analysed events and build a detection method for detecting anomalies to that model. Because of that, they usually comprehend a learning phase where the model is adjusted. They tend to originate more false positives than signature based ones, being this one of its main disadvantages.

When it comes to the source of events, an IDS can be classified as an *Host based Intrusion Detection System (HIDS)* and as a *Network based Intrusion Detection System (NIDS)*. An HIDS is an IDS that works on hosts, analysing the traffic on their network interfaces, system logs and files in general. AVs are within this class. A NIDS is installed and configured in such a way that the environment to be analysed is the network and not specific systems.

In terms of the timing of detection, an IDS can be classified as a (i) Real-Time, (ii) Near Real-Time and (iii) Offline system. The real-time IDS is able to detect attacks and intrusions exactly at the time that they are occurring. In other words, they may be used to take actions in time to prevent the renaming part of the attack. This way of functioning is hard to achieve in most cases. Near real-time systems are the ones that enable an administrator to react to a second incursion by providing timely detection. On the other hand, the offline IDSs only produce results in specific moments (e.g., during the night), by analysing logs or traces captured in other time

periods.

In terms of Reactivity, an IDS may be either passive or active. A passive IDS only produces alarms and reports to inform the system administrator of detected menaces. An active IDS, in addition to the alarms and reports, is able to defend or counter attack the threats also. The defence is performed by isolating the compromised components, by reducing the network bandwidth to mitigate flooding attacks, and by blocking source IPs on firewalls.

Regarding the distributivity, an IDS can be classified as single or cooperative. A Cooperative IDS is an IDS that cooperates with other IDSs to improve the detection performance. This kind of IDSs have the advantage of providing a better understanding on how an infection started and spread. This is due to the fact that cooperative IDSs have sensors that are displaced on several interesting points of a network, making it possible to detect points of entry. Despite these advantages, there is still no standard on how IDSs from different manufacturers can communicate, which is a disadvantage when people and companies wish to deploy different IDSs on their system or network. Single IDSs perform all the steps of the architecture in a standalone mode [Com10, New09, Ina14].

4.2.2.3 IPSs

IPSs come as a natural evolution of IDSs, which kept being improved since the late nineties. While IDSs are mostly passive, only throwing alarms when a menace is detected, an IPS is able to automatically take other actions favouring the system security. An IPS is able to, not only identify, log and throw alarms of attacks and intrusions on the system, as to automatically implement defensive measures, like isolating the compromised components, reducing the network bandwidth to reduce the flooding attacks, and by blocking source IPs on the Firewall configurations. These systems ease some administrative tasks, namely in turns of configuring other systems in the protected perimeter. Since an IPS is more active than an IDS, false positives pose a bigger problem, because, in such cases, the system may be automatically blocking or taking other actions based on a wrong assumption. This is one of the major disadvantages of these systems. If the measure includes blocking essential traffic to a company, the IPS may be a problem bigger than the menace. Many companies prefer using and fine-tuning an IDS first, due to this, to then migrate to an IPS [ICS11, New09, Com10, SM].

4.2.3 Antivirus Software

An AV is a software developed to analyse data, and help the user to protect is machine from unwanted and potentially dangerous threats, namely malware. AVs uses three main techniques to detect and analyse threats [Kas10], described in the following subsections.

4.2.3.1 Signature-Based Analysis

On this approach, the AV examines the file and checks its dictionary or database. If a piece of code on the analysed file matches any virus contained on that data structure, then the AV will flag that file as a malicious program and will proceed with either the deletion or the quarantine of the file or the removal of the virus itself from the file. This approach has a few drawbacks. For example, it requires constant updates to the virus database. These databases can be populated by entries given by technical users, that send their infected files to the AV programmers, who then include that information in the databases. Typically, AV have a process called real-time protection, that runs on the background of the OS, which analyse the files that are created, opened or closed by the OS. Some AVs may also examine the files that are e-mailed or received, prior to being opened. This kind of software may also be scheduled to analyse all the files on the computer file systems by demand of the user on a regular basis. In spite of their advantages, they also have some problems, namely:

1. Malware developers are always trying to improve their tools, and to stay ahead of AVs. Polymorphic malware exists because of this. Some malicious programs contain encrypted parts that are decrypted during execution and re-encrypted or modified before further spreading. Resorting to this way of functioning, malware avoids being detected by the AV because the signatures are different from the ones on the database.

2. The second major problem is related with the previous one. The abnormal growth of the number of malware variants, instigated by the constant search for producing samples that are not detected using current signatures, the databases are growing very fast. It becomes difficult for AV companies to maintain their databases updated. It is also increasingly difficult to maintain record of all samples. Actually, old malware variants have been able to infect computers recently because vendors started to take old signatures out, so as to include the new ones.

Due to the previous points, nowadays, AV companies are starting to consider this option secondary, and are starting to bet on a different approach based on the behaviour. This approach is described in the next section [Kas10].

4.2.3.2 Behaviour-Based or Heuristic Analysis

As the name indicates, this type of analysis, is used to detect potential threats by analysing the behaviour of the programs and files executed by the OS. The quality of this analysis dictates the main difference between a good and a bad AV. A good behaviour-based analysis algorithm can detect threats without having any information of that threat in its database, just by analysing its

behaviour. However, this feature has a few drawbacks, related with false positives and malware unpredictability. Just by looking at the behaviour, the chance to throw false positives is larger. The AV can flag a safe and reliable software or file, as malware, just because it behaved inappropriately according to a predetermined rule set by the AV company. This happened with the Chrome browser some years ago, when Microsof accidentally tagged it as malware [Lie12]. This situation happens because even legitimate and reliable software may behave inappropriately in some situations. Malware can be more unpredictable than legitimate programs. This fact has led the AV companies to develop a set of rules to detect this kind of threats. However, with the proliferation of malware, it became more complicated to predict or model its behaviour. Due to this proliferation, it becomes necessary to also keep updating the respective databases, so that the AV can keep performing well with the new malware trends. Though the database does not grow as fast as for signature-based AVs, they are also getting increasingly more complex.

4.2.3.3 Cloud-Based Analysis

Cloud-based analysis is, according to professionals, the future of AV technologies and some solutions levering this computational paradigm are already available [Tay14]. These AVs perform most of their analysis on remote datacenters rather than on the computers of the users, making them computationally lighter on end-devices.

To explain how the cloud-based analysis works on these AVs, it is important to understand the way this software is implemented. Cloud AVs consist of two main components: (i) the client application and (ii) the server application. The client application is installed in the computer of the user, and has the responsibility to track and report the system activities to the server application. On the other hand, the server application is responsible for maintaining the communication between the client and the server, and to analyse the users files and activities, reporting problems. The communication of these two components is typically done via webservices.

Since the processing efforts are transferred to the server side, the database needs to be on that side also. However, this database, just like for the usual standalone AV is constantly and more easily updated, since it is centralized. So there are a couple of techniques that these servers use to make their databases consistent with the trends.

A Cloud based AVs functions as a proxy server. In other words, every communication entering or exiting the monitored computer has to necessarily pass through the Cloud running the AV server to be analysed, making it possible to populate the database that way also. One of the main advantages of this technology is specifically related with this situation. A normal AV takes an average of 48 days to keep all its users safe against a new threat. However, in this new approach,

a cloud-based one is able to do it in 6 minutes. These AVs also have the ability to communicate with honeypots, scattered worldwide. Given these facts, it is safe to assume that this technology comes with some advantages over usual AVs. The user does not need to keep constantly updating its database, the analysis of the files, which is a heavy process for the computer processor, does not need to be done locally, leaving it free for other activities. However, there are also some disadvantages. For example, users may not want to have their personal files and programs circulating over the Internet to be analysed in some remote datacenter, of which they may not know the exact privacy policy.

Other big issue is related with the Internet access. Since these AVs need constant communication with the servers, if the computer is not connected, the AV may not function properly. Normally, the client application of this system keeps a local and simplified copy of the database for those cases, containing only signatures for the most severe threats. Since most of malware is spread through the Internet, this may not pose a major threat. These AVs usually use both Signature-Based and Heuristic-Based analysis [Cra, Sec11].

4.3 Dissection Tools

A lot of threats need a malware analyst to try to describe their functioning and identify their purposes. For performing their tasks, they may use several analysis approaches. Typically, their main objective is to isolate all the infected machines and files on a compromised network or system. After that, they are the ones responsible for providing information about the threat behaviours and produce signatures to the AVs companies, or to the administrators, so that the latter can configure the required protection mechanisms. In most of the cases, an analyst is forced to work based only on an executable file, in a non-human-readable language. The two approaches to analyse this file are called *static analysis*, which is performed without executing the malware, and *dynamic analysis*, in which the malware is studied while being executed. These approaches are briefly described in the following subsections [SH12].

4.3.1 Static Analysis

Static analysis techniques are performed without the execution of the malware code. This approach can be further subdivided into two types: the *basic* and the *advanced* static analysis. On the basic static analysis, the malware analyst examines the malware executable without looking at the actual code. Basic static analysis is able to give the analyst information about the threat functionality. However, a sophisticated piece of malware will make this approach useless. In the advanced static analysis approach, an attempt to disassemble the malware sample is performed, potentially providing the analyst, which will have to analyse the code instructions, with more information. This constitutes, normally, a painfully slow and complicated task, requiring the analyst to have knowledge on disassembling, code constructors and OS concepts.

4.3.2 Dynamic Analysis

Dynamic analysis comprises techniques that are used to get information regarding a malware sample by running its executable. This approach can also be further subdivided into *basic* and *advanced* dynamic analysis. In both cases, several tools can be used too observe the behaviour of the surrounding infected system. It is thus common to prepare a safe environment in which the sample runs without jeopardizing sensitive information or critical services. Nowadays, virtual-machines are used to this purpose, leading malware developers to implement anti-virtual machine techniques to prevent the analysis. The most advanced form of analysis comprises using a debugger too, in addition to the aforementioned tools. Usually, this approach is used combined with the advanced static analysis to obtain the most complete amount information on a given sample.

4.4 Conclusion

This chapter gives an overview about good practices that a user may adopt to prevent being infected or spreading malware. Then evolves to the description of systems and software that can be deployed to fight malware, namely firewalls, IDSs, IPSs and, more particularly, AVs. It was emphasized that AVs may be moving to the cloud, taking some of the computational burden from the end-devices, but requiring the potentially private data to travel between the computer of the user to the remote datacenter.

Towards the end of the chapter, the two basic approaches that an analyst may follow to study a new sample of malware were also described. The techniques and tools mentioned in section 4.3 are important in the context of this Ms.C. dissertation, since they refer directly to the dissection of malware. As such, they are going to be explained with more detail in the following chapters, along with the description of analysis made to real samples. The aforementioned section comprises solely an introduction to this part of the work. When a new sample appears, it is the job of the analyst to study it, name it and provide signatures or the behaviours that may enable the detection of the threat.

Chapter 5

Basic Static and Dynamic Analysis

5.1 Introduction

The process of analysing and studying malware has been gaining importance in the last few years. Unfortunately, given the importance of computers and networks nowadays, it is relatively easy to motivate malware development. Studying and analysing malware technologies is necessary, not only to develop defence mechanisms as also to solve ongoing incursions. Producing rules for firewalls, or signatures for IPSs and IDSs is only possible if specialized personel is devoted to that analysis.

This chapter describes the basic static and dynamic analysis processes, but first, software and techniques that are typically used to find information about a specific malware are introduced. It is intended to provide a structured approach to the analysis process and, as such, the chapter is structured as follows. The next section is focused on the aforementioned techniques and tools, while sections 5.3 and 5.5 elaborate on the two main types of analysis. In the middle of those two, section 5.4 is devoted to the setting up of a safe environment.

5.2 Techniques and Tools

This section introduces some tools that are used for the so called basic static and dynamic analysis. It is thus divided into two main subsections.

5.2.1 Basic Static Analysis

Before applying any technique or tool to a threat, it should be tested if it was not already identified and catalogued by an existing AV. As different AVs programs use different signatures and heuristic methods, it is important to run the threat on more than one AV product. To do that, there are several online services that will automatically submit the threat to several AVs, generating a report with the results from all of them. An example of a popular site offering this service is *Virustotal* [tea]. AVs are thus one of the most important analysis tools.

The hash value is very commonly used as one of the signatures associated with a certain malware, and can be used in various ways. For example, it can be used as an AV signature, sent to another malware analyst so that two files can be compared, or to search online for information on that sample. Another usual application of the hash value is to label the threat, before giving it a

proper name. Several algorithms can be used for this purpose, namely Message-Digest algorithm 5 (MD5) and Secure Hash Algorithm 1 (SHA-1), with MD5 being the most popular one. MD5 and SHA-1 values can be obtained on the *Virustotal* website [tea] too, however, it is possible to use a standalone program that easily calculates them. In a Microsoft Windows OS, the MD5 can be easily calculated using the `WinMD5` utility [Win].

During execution, programs show certain messages and information, usually hardcoded in the program. In the malware related jargon these messages are known as *strings*. When found in malware samples, they may actually provide crucial info regarding their behaviour. For example, if the program communicates with an external server, the IP address must be included in the code. If the threat creates or moves a file then the filenames may also be within the code in the form of strings. It becomes crucial to analyse the strings of a suspicious program when trying to establish its maliciousness. A way to analyse these strings is by using a software like `Strings` [Rus13], a well known CLI application that extracts all the strings from a program. The process of extracting strings is not simple, because an executable is not in ASCII code, making it hard to distinguish processor instructions and other information that might be garbage to the analyst. The analysis of the returned strings is then of the responsibility of the analyst.

Often malware threats are either packed or obfuscated, sometimes both. A common feature of obfuscated or packed programs is that they typically have a very small number of strings, when compared to normal programs. To check if a file is either packed or obfuscated, a program like *PEiD* [Sna08] can be used. Before proceeding with the analysis, it is a good practice to check if the file is either packed or obfuscated. If the file is packed, it must be unpacked before proceeding. Packing and obfuscating techniques, as well as the unpacking process are further explained in Appendix A A.1.

In order to implement the required functionalities, programs often use dependencies like libraries and functions. The explanation of library linkage can be found in Appendix A A.2. The libraries and functions invoked by programs can often be used to induce some behaviours. By analysing the linked libraries, the analyst may be able to discover some or all of the features of the threat. *Dependency Walker* [Mil06] is a software which allows the analyst to check the linked libraries and functions. Because the analyst may not be familiar with all the libraries and functions of the OS, i might be difficult to extract the exact meaning of their inclusion in a program. It might help to look for a list of Windows OS libraries and functions known to be used to create malicious programs, like the one included in the book entitled*Practical Malware Analysis* [SH12].

Until this moment the techniques and tools described where only able to obtain some of the most direct information of a threat, and did not look at its structure. However, it is possible to extract information from the format of the file itself. Because the scope of this Ms.C. programme is

confined to the Microsoft Windows OS family, the PE file format is the most interesting one. A discussion concerning the structure of this file format is therefore contained in Appendix A A.3. The analyst may use *PEView* [PEV14] to investigate PE files headers.

Similarly to PEView, *Resource Hacker*, a software used to analyse the resource section of the PE files, is also focused on the file format. It can graphically show the icons that the threat will use, Dialog Boxes, Strings and the Menus that the threat will exhibit during execution, etc.

5.2.2 Basic Dynamic Analysis

There are several available software solutions allowing the analyst to automate the basic dynamic analysis process. These solutions might be used to speed up the analysis and, most of them, rely on *Sandboxing technology*. A *Sandbox* is a security mechanism created to run untrusted programs in a safe environment. Sandboxes are usually composed by a virtualized environment, which often is able to simulate some network services, with the main objective of allowing the malware to function normally, even when requiring files or data from an online resource. Most of the available sandboxes for desktops, that can automatically analyse malware, are expensive and, as such, the best option may be to use an online sandbox like *Malwr* [CGN], which relies on the nowadays popular open-source project called *Cuckoo Sandbox* [Fou14a]. Cuckoo can also be installed on the local machine through a still complicated process, which can only make sense if the analyst suspects that the malware sample under analysis contains some important information that should not be transmitted over the Internet. However, if that is not the case, *Malwr* will provide all the results that could be obtained from the cuckoo sandbox.

simulator used by the sandbox will not know exactly what message the malware will require to continue its execution; enough; and *Process Monitor* allows monitoring the processes that are being executed on the machine on which the analysis is being performed. The objective ist to keep track of all the activities and actions that the processes may perform. It can also be used to track some changes to the system registry, the file system, the network, the processes it creates or interacts with, and the thread activity [RC14b]. Process Monitor is not able to capture certain device drivers activities, like communications between a device and a rootkit via Input/Output controls, as well as some GUI calls. Despite being able to monitor network communications, it should not be used for that purpose, because it was not tested in different Microsoft Windows versions. If the analyst wants to monitor network traffic, it is better to use the *Wireshark* tool, explained later. As an OS may have more than 50000 system calls in one minute, it becomes a matter of practice and patience to identify the important information that can be gathered with the usage of this tool. For that, the usage of *filters* is crucial and should be used to keep the amount of data manageable.

Process Explorer is a SysInternals tool, similar to the Windows Task Manage of Microsoft Windows

OSs [Rus14]. Contrailly to the latter, it allows the analyst to view other information about processes currently running on the system. This is one of the most useful tools in dynamic analysis. Process Explorer is typically used to list all active processes, loaded DLLs and some key process properties. It makes it possible to obtain the process hierarchy, the process signature (if the process is signed with any asymmetric cryptography mechanism), the process timeline (the line of time representing the execution of the process since it was started). It also provides information about the Transmission Control Protocol / Internet Protocol (TCP/IP) connections, established by each specific process, and provides the strings included in the process. It allows the malware analyst to identify the new processes, as well as the recently terminated ones, highlighting them in different colours. Another useful feature of Process Explorer is the ability to compare in memory strings with the ones saved on the persistent storage. This is useful for when malware is injecting code into another process.

AutoRuns was developed by Windows SysInternals and allows the users to see all the autorun entries on the OS [RC14a]. AutoRuns might be used to find changes on the system registry and to detect the autorun entries on Microsoft Windows OS. It is one of the most complete startup monitors that can be found online, and has the most cohesive knowledge of Microsoft Windows OS auto-starting locations. Due to that, it is considered an important tool when it comes to find possible maliciously created autorun entries.

The system registry is a very important component of the Microsoft Windows OS. It stores data regarding the configuration of the computer. It includes information on user accounts, installed programs and types of documents, along with the association between them, properties of folders and icons, driver configurations, and the programs that should autorun at the system log on. By investigating the new entries that an executable might create, it is possible to identify some information about the malware.

Regshot [Cor14a] functioning is rather simple. It takes two snapshots of the registry: one taken before the malware is executed; and the other one after.

Most of the malware samples nowadays use a network connection to deliver their full payload. As such, it is important to simulate a network or at least provide a real, but controlled, connection. If the analysis can be conducted using a fake network, then this should be the preferred option. Otherwise, care should be taken when setting up a real connection.

Apate Domain Name System (DNS) is a software that redirects DNS requests to the Internet simulator (e.g., *InetSim* - see below) running on the virtual machine [Man14]. The Apate DNS is set with the IP of the virtual machine and when a DNS request is made by the host OS and, in the context of malware analysis, by the malware sample, Apate DNS spoofs the DNS response and answers with the IP of the virtual machine.

Wireshark is the most popular packet sniffers [Fou14b]. It provides a GUI where the traffic is displayed in real-time after being parsed up to the application layer protocol of the OSI model. As such, it is ideal to study the communications of malware.

INetSim is a free and widely used tool for basic dynamic analysis. It is used when the network is not available or simulated in a larger laboratory environment [HE14b]. The main objective of INetSim is to simulate some of the most common Internet services, namely HTTP, HTTPS, FTP, IRC, DNS, Simple Mail Transfer Protocol (SMTP), Post Office Protocol 3 (POP3) and a few others. Sometimes, INetSim enables the malware to keep functioning and fetch fake files from the simulated environment so that the analyst can collect more information.

INetSim can only be installed on a Linux OS. When mentioned below, this remark should be taken into consideration. To simulate the network on the virtual laboratory used in the scope of this work, a virtual machine with Xubuntu was setted up. This is illustrated in figure 5.10.

5.3 Basic Static Analysis

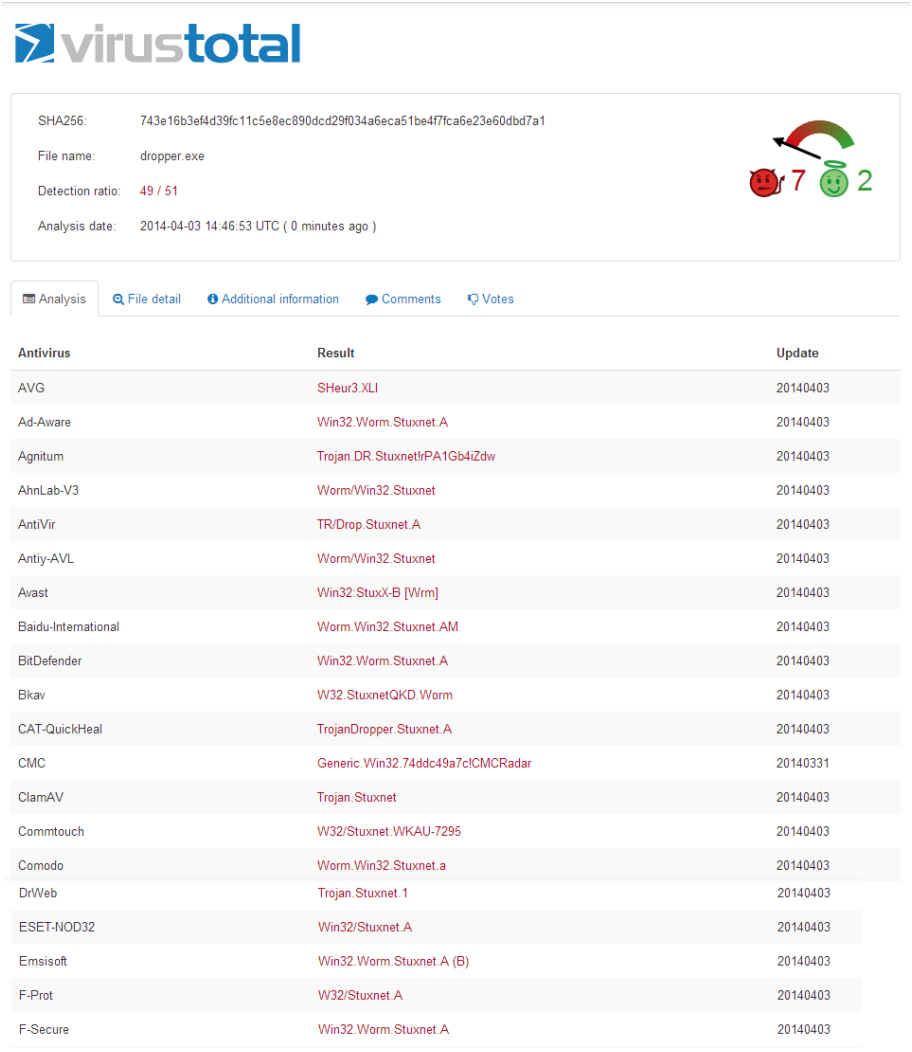
After having described the techniques and tools used to perform the basic static and dynamic analysis, it is important to explain each type of analysis. The approach followed by analysts is introduced. The first step is to apply basic static analysis techniques. Only then the installation and correct configuration of a safe environment is suggested, since it provides the analyst with the means to advance to the basic dynamic analysis phase. This process is not standardized, and it results from the overview conducted over the literature on this area.

Basic static analysis allows the analyst to extract key information that may be useful to select what to do next. One of the big advantages of starting from this step is the ability to prepare better for what to use afterwards, without needing that much knowledge about the threat and running the malicious executable on the computer.

Some key information gathered in this phase is comprised by some behaviours and the interface type of the threat. To exemplify the application of this approach, a sample of a variant of the `Stuxnet` worm was downloaded and studied in the scope of this Ms.C. programme. Below, a brief report on this experiment is included. The steps taken to analyse the sample of `Stuxnet` where based on the book entitled *Practical Malware Analysis* [SH12].

`Stuxnet` is a worm known worldwide and commonly detected by most of the modern anti-malware solutions. However, it is still considered one of the best and most sophisticated pieces of malware ever created and found in the wild.

All the techniques and tools used in the course of the analysis where been described in the previous chapter. The first step was to find if the supposed threat has been previously flagged as



virustotal

SHA256: 743e16b3ef4d39fc11c5e8ec890dcd29f034a6eca51be4f7fca6e23e60dbd7a1

File name: dropper.exe

Detection ratio: 49 / 51

Analysis date: 2014-04-03 14:46:53 UTC (0 minutes ago)

7 2

Analysis | File detail | Additional information | Comments | Votes

Antivirus	Result	Update
AVG	SHeur3.XLI	20140403
Ad-Aware	Win32.Worm.Stuxnet.A	20140403
Agnitum	Trojan.DR.StuxnetIrPA1Gb4iZdw	20140403
AhnLab-V3	Worm/Win32.Stuxnet	20140403
AntiVir	TR/Drop.Stuxnet.A	20140403
Antiy-AVL	Worm/Win32.Stuxnet	20140403
Avast	Win32:StuxX-B [Wrm]	20140403
Baidu-International	Worm.Win32.Stuxnet.AM	20140403
BitDefender	Win32.Worm.Stuxnet.A	20140403
Bkav	W32.StuxnetQKD.Worm	20140403
CAT-QuickHeal	TrojanDropper.Stuxnet.A	20140403
CMC	Generic.Win32.74ddc49a7c!CMCRadar	20140331
ClamAV	Trojan.Stuxnet	20140403
Commtouch	W32/Stuxnet.WKAU-7295	20140403
Comodo	Worm.Win32.Stuxnet.a	20140403
DrWeb	Trojan.Stuxnet.1	20140403
ESET-NOD32	Win32/Stuxnet.A	20140403
Emsisoft	Win32.Worm.Stuxnet.A (B)	20140403
F-Prot	W32/Stuxnet.A	20140403
F-Secure	Win32.Worm.Stuxnet.A	20140403

Figure 5.1: Results of the Virustotal analysis on the `Stuxnet` malware sample.

malware on any antivirus or anti-malware solution, either locally or online. It becomes crucial, as previously explained, to run the sample through multiple anti-malware solutions. In the context of this work, the *VirusTotal* website was used to perform this first task, and the results of the multiple anti-malware solutions for the `Stuxnet` sample are presented on figure 5.1.

As can be seen in the figure, there are several anti-malware solutions that flag the suspicious file as malware. However, as previously emphasized in Chapter 3, not all of them give the threat the same name. There are also two anti-malware solutions flagging the file as not malicious. These results were expected given the fact that, today, most of the anti-malware solutions already have information to detect the `Stuxnet Worm`. Nonetheless, on a real context of malware analysis, new threats may not be detected by anti-malware solutions. This is more like an initial standard procedure.

The next step on malware analysis, and if the information obtained previously was not conclusive, is to use an hashing algorithm to calculate the hash value of the file. As the mostly used

hashing algorithm on malware analysis is the MD5, figure 5.2 shows the output of the WinMD5 program when fed with the `Stuxnet` sample.

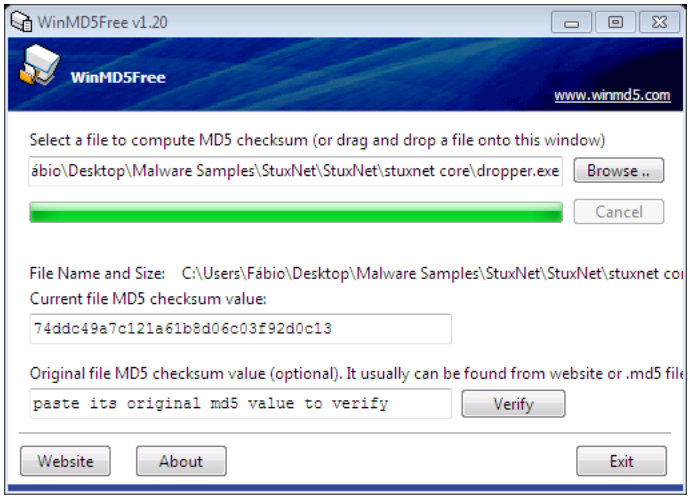


Figure 5.2: WinMD5 result on the `Stuxnet` malware.

The tools and techniques applied up to this phase only provide very general information regarding the malware sample, and where not based on the analysis of the threat by the analyst per se. The next step is to analyse the strings. Often, a malware sample uses strings in order to store information that it may need. Windows Sysinternals *Strings* was used to analyse the strings of the `Stuxnet` PE file. Figure 5.3 shows the results returned by this tool with the option to only find strings that have 4 or more letters on the `Stuxnet` PE file.

The objective now is to draw some conclusions about the observed strings. Most of them will not provide any valuable information, so they might be discarded. However, some will really be of great value to understand some of the basic functionalities of the threat. The most important strings found during this phase have been highlighted using red and blue boxes in the previous figure.

The strings surrounded by the blue squares are related with sections of the PE file header. The `.text`, `.rdata`, `.data` and `.reloc` do not convey any particular meaning. However, the `.stub` section may indicate that this malware has a *stub* which, in most cases, suggests that the malware is packed. The main conclusion is that static analysis without unpacking the PE file is a difficult process that may probably lead nowhere. The stub program is the only artefact that can be readily analysed in this situation. The strings surrounded by red squares are probably imported functions by the threat to deliver its payload. The most important of this set of strings are described below:

- `GetProcAddress`, which is used to import functions from other DLLs in addition to the ones imported at runtime, and may indicate that the program uses a dynamic linking technique;

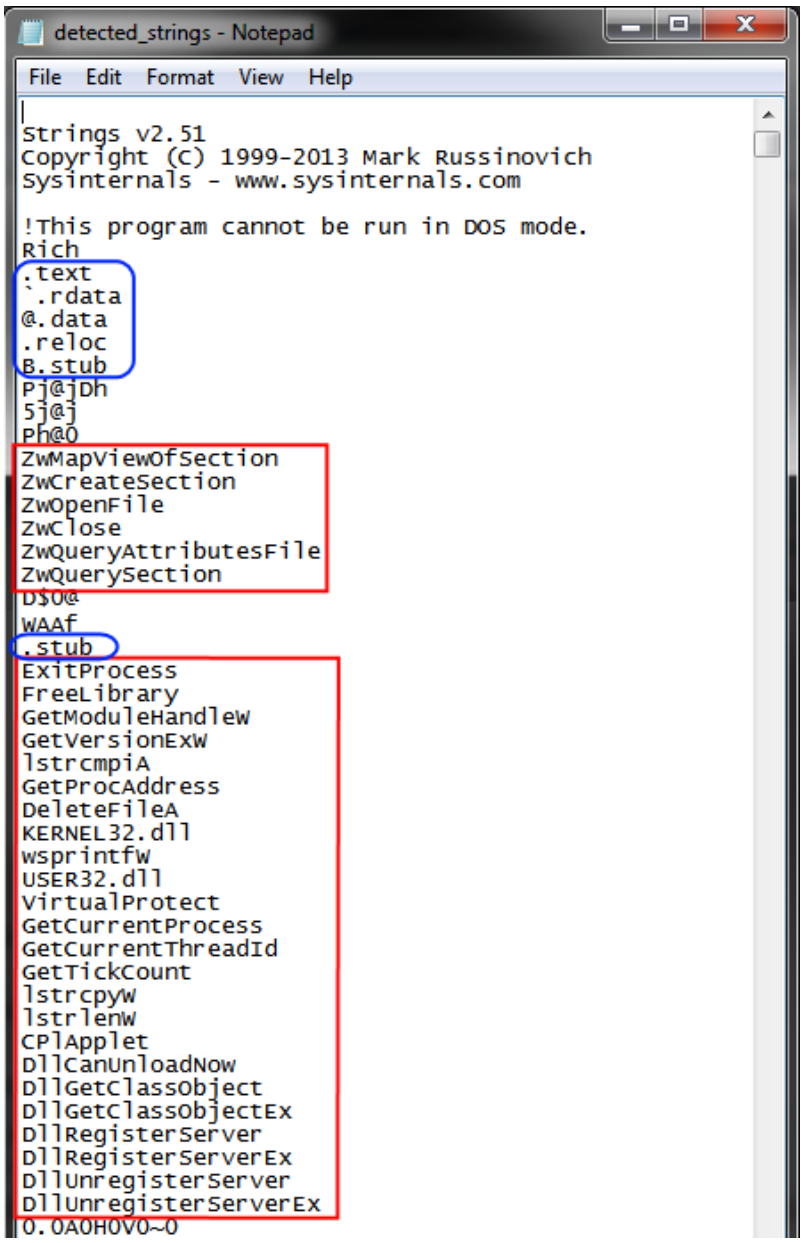


Figure 5.3: The result of applying the Strings tool to the `Stuxnet` malware sample.

- `GetModuleHandleW`, which is commonly used by malware to find a good place to inject its code. Given the fact that malware may benefit from being hidden, this function is commonly invoked when it tries to remain hidden;

- `DeleteFileA`, which is used to delete files, and malware may invoke it to erase its trace.

Now that the strings are analysed, the next step is to confirm if the file is either packed or obfuscated. To do that, the *PEiD* software was used. Since it is not 100% accurate, it might not detect any packing or obfuscation and still the file might be packed. The problem is that concealment algorithms are getting better and constantly evolved. In the of the `Stuxnet` sample, PEiD did not detect packing. The results of the PEiD `Stuxnet` malware sample might be seen in

figure 5.4.

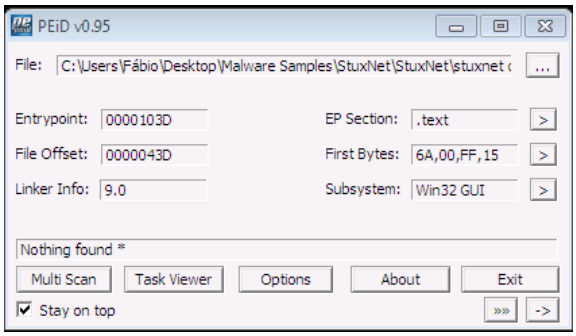


Figure 5.4: Screenshot with the results provided by PEiD when analysing the `Stuxnet` malware sample.

The PEiD software also tells us what is the program subsystem. A PE subsystem is the environment where the program will execute. It might either be CLI or GUI. In this case, the `Stuxnet` malware sample belongs to the GUI subsystem, which means that the program will exhibit a GUI when running, which must be defined on the `.rsrc` section of the PE.

Following on the basic static analysis steps is the analysis of the imported and exported libraries and functions. Part of this task was already performed when analysing the strings, given the fact that the name of the imported and exported functions where already present on them. Nonetheless, there might functions whose name does not appear in the strings analysis (e.g., the ones whose name is a ordinal). As such, it is important to look at the *Dependency Walker* results, depicted in figure 5.5, which provides the information about the libraries imported and the functions called.

As can be seen in figure 5.5, there are two imported libraries, `KERNEL32.DLL` and `USER32.DLL`. The `KERNEL32.DLL` is the selected library, and the functions that the `Stuxnet` malware sample imports are the ones on the top right pane. The middle pane shows every functions that might be imported from the selected library and information about those functions. Functions might be imported by name or by ordinal number. If the functions are imported by their ordinal number, it becomes necessary to search on the middle pane for the corresponding function. It might be hard to find some imported functions on the middle of the strings previously obtained. The bottom two panes only list information of the DLL that is selected and errors that might be presented when executing the malware sample. Finished looking at the imported and exported functions the next step of a malware analyst is to look at the PE file headers to try to obtain more information on the threat.

As can be seen in figure 5.6 the PEView software gives information about the PE file showing, on the left pane, the most important parts of the PE header and, on the right, the contents of the currently selected section. The two first sections on the left pane `IMAGE_DOS_HEADER` and *MS-DOS Stub Program* do not offer any particular information to the analyst and, as such, they

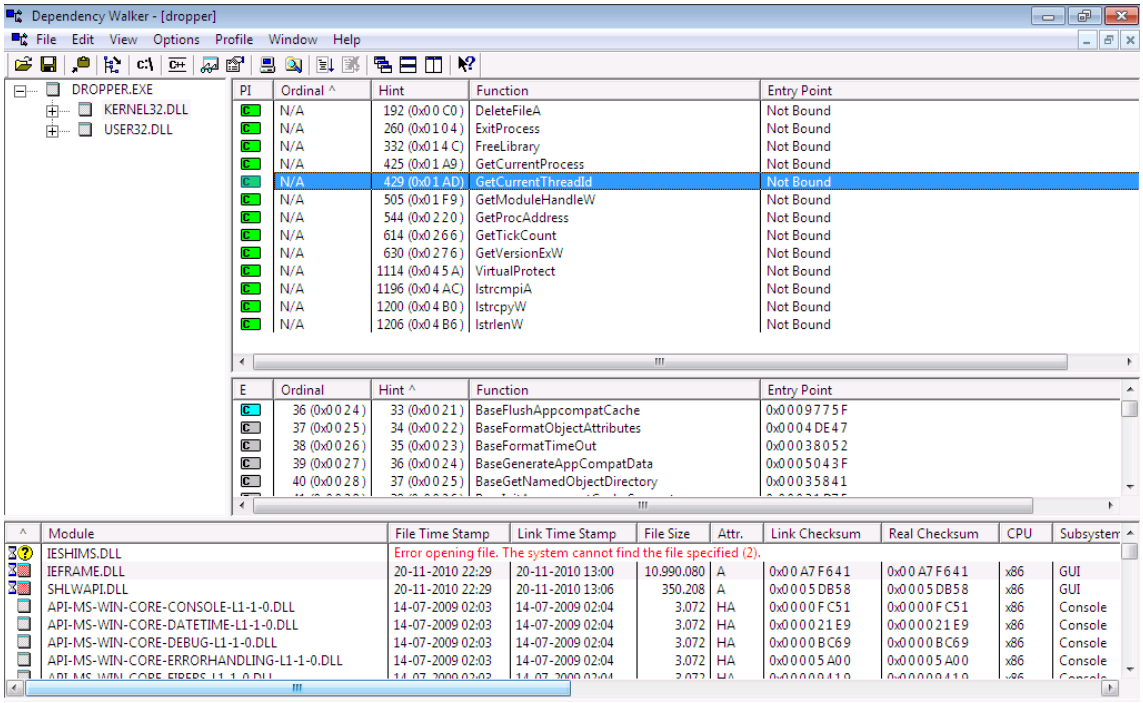


Figure 5.5: Screenshot of the Dependency Walker application, showing the libraries imported by the `Stuxnet` malware sample.

are not shown nor explained herein.

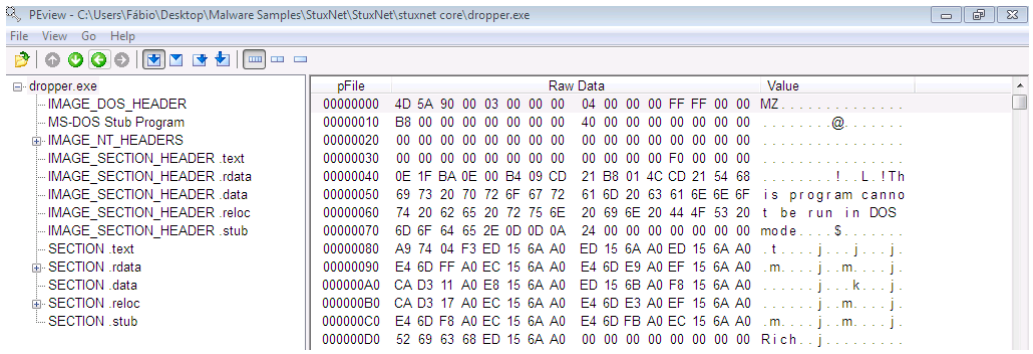


Figure 5.6: Screenshot of PEView when analysing the `Stuxnet` malware sample.

By analysing the section `IMAGE_NT_HEADER`, it is possible to find, the compilation time of the executable, under the *Time Date Stamp* field, like depicted in figure 5.7. However, this information is not reliable, since a malware developer may change it, making it impossible for an analyst to temporize the threat.

Since the `IMAGE_OPTIONAL_HEADER` might hold some valuable information too, namely regarding the PE file subsystem, it was also analysed. The `IMAGE_OPTIONAL_HEADER` of the `Stuxnet` malware sample is presented in figure 5.8.

The `IMAGE_SECTION_HEADERS` contain information regarding the size of the programs in disk (size of raw data) and in memory (virtual size of data). The size of raw data and the virtual size of

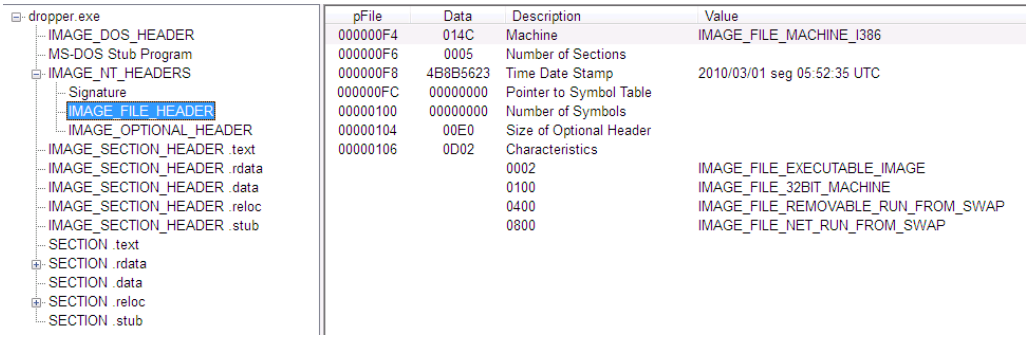


Figure 5.7: Screenshot showing the `Time Date Stamp` field when analysing the `Stuxnet` malware with PEView.

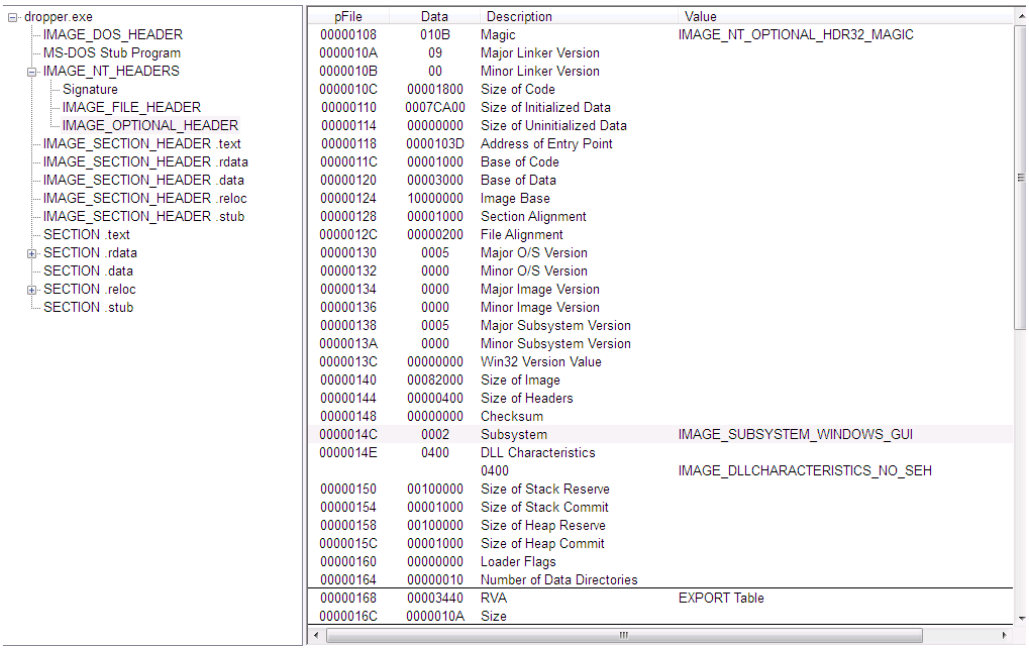


Figure 5.8: Screenshot showing the contents of the `IMAGE_OPTIONAL_HEADER` of the `Stuxnet` malware.

data should be more or less the same, because data should take the same space on disk and on memory. Small differences between those two values should not be taken into account either, since the data is not stored the same way on both memory systems. Nonetheless, by looking at the two values the analyst might be able to tell more precisely if the threat is packed. This can be concluded when the virtual size is much bigger than the raw size of data. If this occurs for the `.text` section, then one may more confidently conclude that the threat is packed. This is not typically applied to the `.data` section.

Figure 5.9 depicts the `.text IMAGE_SECTION_HEADER`. By analysing the figure, it may be concluded that the file is not packed. However it might still be interesting to study the `.stub` section in a more advanced stage.

The next stage is to look into the `.rsrc` section with the *Resource Hacker* tool. The `Stuxnet` sample that was tested in the scope of this work did not had any resource section, thus not

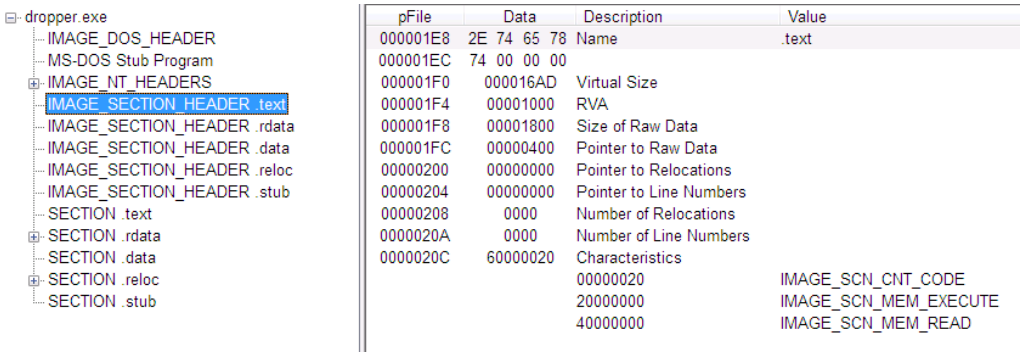


Figure 5.9: Screenshot showing the contents of the `IMAGE_SECTION_HEADER` of the `Stuxnet` malware.

providing any further useful information.

The previous procedure shows that basic static analysis is only the beginning. It is used to assess if the sample runs via a CLI or if it has a GUI, to infer about potential behaviours of the program, get linked libraries and conclude if obfuscation or packing was used.

The analysis performed on the `Stuxnet` malware sample, returned no particularly interesting information. It was a good example to start with, but will be replaced with another malware sample afterwards. For this particular sample, it was only concluded that it was not packed and that it might try to delete files to hide in the system, along with loading more code into memory while executing. Nonetheless, when submitted to Virustotal it was flagged as malicious.

The next phase consists in the basic dynamic analysis, which involves running the malware sample and study its behaviour in the meanwhile. Nonetheless, there are a few things to consider before proceeding into that phase, namely regarding how to build up a safe environment to run the samples.

5.4 Setting Up a Safe Environment

In order to install and configure a safe environment, it is important to consider a few factors. Most of the professional malware analysts have a dedicated laboratory with a few computers configured on the same network, although not typically connected to the Internet, and with different characteristics and features, specifically designed to test and detect all the malware activities. During this Ms.C. programme, a dedicated laboratory for testing malware was not available and, as such, the alternative was to run the malware locally on a dedicated, and isolated physical or virtual machine.

Some malware samples might detect that they are being executed on a virtual environment. These anti-virtual machine techniques need to be considered when building the safe environment. Nonetheless, the usage of virtual machines has the advantage of being easy to reset the settings and provide the commodity to run the malicious software in a fully sandboxed environ-

ment. On the other hand, by running the malware sample on a physical machine, anti-virtual machine techniques become useless. Another important point that needs to be considered when running the malware on a physical or virtual machine is the Internet connection. Some malicious programs need a single access to or an always-on Internet connection to deliver their full payload.

Figure 5.10 shows how the safe environment was created to perform the dynamic analysis during this Ms.C. programme. The desktop computer used to implement the aforementioned environment was a Fujitsu Siemens Computer, running Microsoft Windows 7 Professional Service Pack 1 32 bit OS, with and Intel Pentium D 3.00GHz Central Processing Unit (CPU), 1 GB of Random Access Memory (RAM) and a 300 GB Hard Drive. The VMWare player was installed on top of the Windows 7 OS, and used to install the virtual machine. A very light Linux distribution known as Xubuntu 4.10 was then installed on the virtual machine. The Linux machine was used to install the network simulator. The configuration of INetSim was performed as suggested in its official website [HE14a].

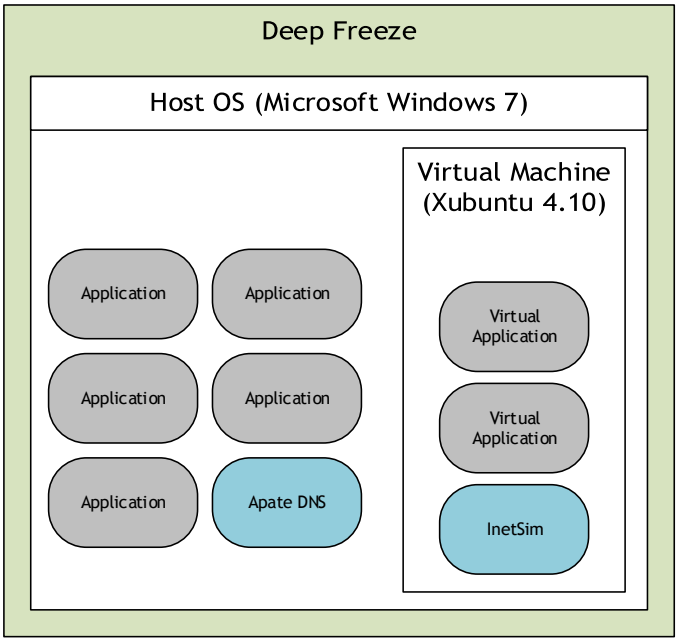


Figure 5.10: Technologies and tools used to build the malware testing safe environment.

Afterwards, a software called *DeepFreeze* [Cor14c] and *Data Igloo* [Cor14b] were installed on the Windows OS. DeepFreeze is a tool created to help the preservation of the desired configurations of a computer. Once configured, the DeepFreeze software, allows only for temporary modifications on the protected computer. Once the computer is restarted, all the modifications performed are reversed. DataIgloo offers a way of keeping a few files saved even after the restarting of the computer. It is the only way to keep files that where created or modified with the DataFreeze active.

After all these configurations where performed, there are two machines within the same physical one (a host and a guest system). It should always be confirmed that there is no Internet connection, e.g., a wired connection to the network may automatically configure IP addresses. The simulation of the Internet was solely performed using INetSim and the ApateDNS. *ApateDNS* has been configured so as to answer to all the DNS requests with the IP address of the virtual machine running *InetSim*. With this configurations, all of the communications established by the computer are controlled and answered by the virtual machine network server simulator.

Network communications within this safe environment can be further explained resorting to figure 5.11. As depicted in the figure, if a new communication is initiated, the DNS request is sent to the ApateDNS tool (communication A), which then answers with the IP address of the Virtual Machine (communication B). INetSim responds to the malware request with the desired type of information, if the requested and resource is available (communication C). The simulator

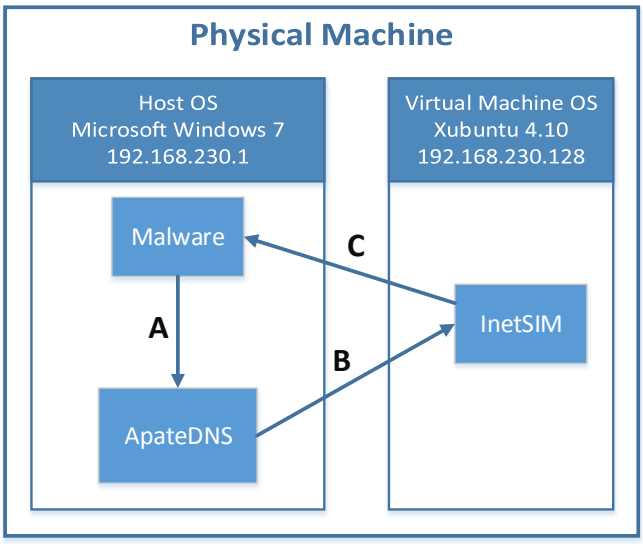


Figure 5.11: Scheme representing network communications within the safe environment.

may not correctly emulate the environment with the files or variables that the malware requires to continue its correct operation. Nonetheless, if the malware requires for a simple media file, like for instance, a `.jpg` file, the simulator may provide it. If a more specific file or string is requested, then it will probably fail.

5.5 Basic Dynamic Analysis

Basic dynamic analysis is the second main phase of malware analysis, and provides a way of obtaining information from the execution of the threat. Despite the fact that the computer in which the tests are performed ends up infected, it is a good and simple source of information, even with the inherent need of setting-up a good safe environment. Once the safe environment is configured, the malware may be executed on the computer. However, it should be noticed that different malwares may present different behaviours when executed. It is normal for the

computer to act strange during this type of analysis. Basic dynamic analysis is able to provide more cohesive information about the malware functionalities, since all of the actions the sample performs can be monitored and analysed while running.

A total number of 15 variants of the `Stuxnet` worm found online were tested within the scope of this work. Unfortunately, they all seem dormant and did not produce significant results worth of discussion beyond what was said in the previous sections. The reasons for this to happen may have been related with the environment in which the samples were analysed, which could already contain patches or missing information that the worm requires. Alternatively, it may also be related with the sophistication of this malware.

It was then decided to perform the basic and advanced dynamic analysis with the `Tinba/Zusy` malware sample, which is also relatively recent. As the first step of analysis was already documented above resorting to the `Stuxnet` sample, the basic static analysis concerning `Tinba/Zusy` is described in Appendix B B.

The first step of this phase is then to get a registry snapshot before proceeding with the execution of the sample. Malware, as well other typical programs, may create, alter or delete Windows registry entries. Since the registry is responsible for many tasks when it comes to the OS control, it is important to obtain a clean copy of the registry before proceeding with the malware piece analysis, since it may latter contain indications about actions and behaviours of the analysed sample. To perform a registry snapshot, *Regshot* was used, of which an example is shown in figure 5.12.

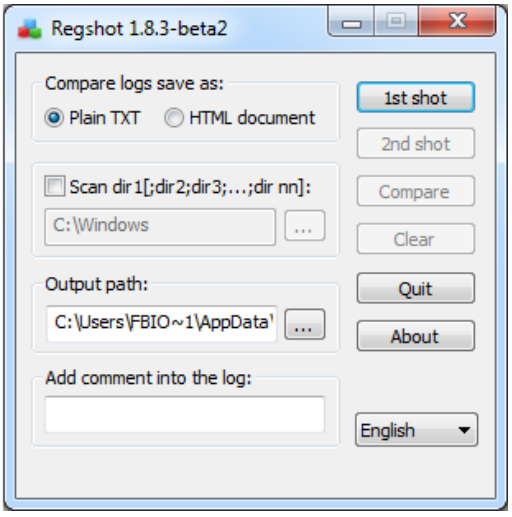


Figure 5.12: Screenshot of the *Regshot* application while executing on the system.

Further information of the Windows Registry can be found in Appendix A A.5. Once the snapshot of the system registry is performed, it is necessary to find a way to monitor the process activities on the system. Two tools that provide these functionalities are ProcMon and AutoRuns.

Once executed, AutoRuns searches the computer for autorun entries on all of the Windows OS predefined autorun locations, retrieving a list of the existent autorun entries, just like depicted in figure 5.13. This software does not operate in real time and, as such, before running the sample and start looking for its behaviours, it is necessary to a save the previously obtained entries. Similarly, after executing the malware, it is necessary to repeat the autoruns search. Once the two AutoRuns results are obtained, they are both compared in order to discover new entries.

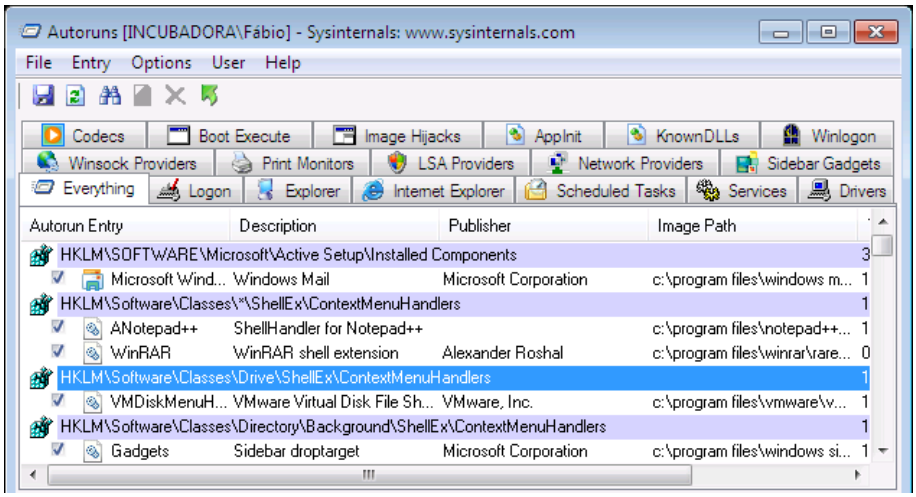


Figure 5.13: Screenshot with the results retrieved by the *AutoRuns* tool.

ProcMon displays all the activities of the currently opened processes, as shown in figure 5.14. In order to allow for a better perception of the malware sample activities, some filters might be applied to the displayed activities. ProcMon offers the possibility of filtering the activities based on lot of different characteristics. Most of the processes which are usually displayed belong to the OS itself. Even despite malware can inject code into some processes (Process Hijacking), the analysis usually starts by focusing on the main process of the malware sample.

The next step consists on starting the *Process Explorer* tool to subsequently try to discover the main process name and Process Identifier (PID) (if any is created during the execution of the malware). Finally, before executing the malware piece, it is necessary to start *Wireshark*, so that any communications can be traced and analysed. Once all the aforementioned tasks and configurations are performed, it is time to execute the malware piece.

After the execution there are a few actions that need to be performed, namely but not precisely in that order:

1. Try to discover the process created by the malware piece;

2. Take another snapshot of the system registry;

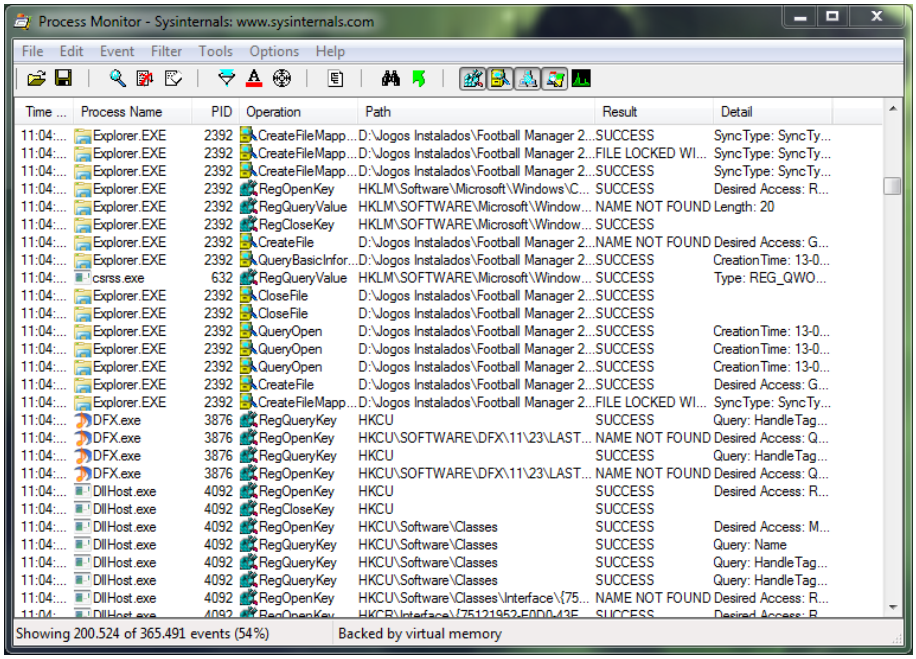


Figure 5.14: Screenshot of the *Process Monitor* application with the default activities filters.

3. Save the results of the *ProcMon* in a file for further analysis;

4. Save Wireshark results;

5. Analyse network activity provided by the ApateDNS software;

6. Check the AutoRuns result after refreshing;

7. Restart the machine to prevent any misrepresentation on the results.

Analysis resumes by analysing the information obtained so far.

After the execution of `Tinba/Zusy`, the Process Explorer tool was consulted, and the name of the created process was obtained. In this case, as can be seen in figure 5.15, the name of the new process is the one highlighted in green and the PID was 2876. With this information, it is possible to efficiently apply filters in the ProcMon tool.

And on top of this information, it is possible to observe that a strange and unfamiliar DNS request was made during malware execution in ApateDNS. ApateDNS answered to a request with the `dakotavolandos.com` URL. Nonetheless, one can only know what is being requested for sure after analysing the Network Traffic generated and captured by Wireshark (see figure 5.16), which comprises the next logical step.

During the analysis of the captured packets, it was noticed that several attempts to communicate with the `dakotavolandos.com` server were performed, requiring a Hypertext Preprocessor (php)

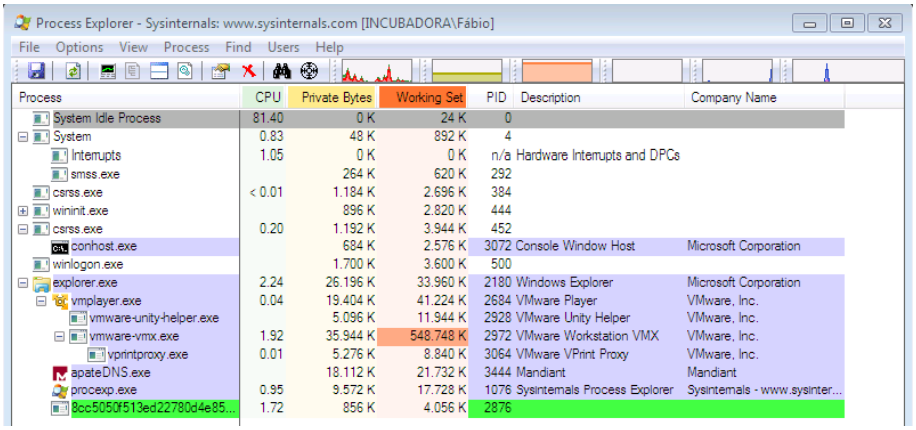


Figure 5.15: Screenshot of the *Process Explorer* program showing the Microsoft Windows process list.

script to be run. One thing that can be said is that this malware sample does not appear to spread on its own through the Internet connection, once it does not make any network communications suggesting that behaviour up to this phase. Nonetheless, an up to this point, one cannot be actually sure the analysed program is, in fact, a malicious software (this conclusion will be drawn latter on, along with the statement that the sample does not belong to a self-replication threat).

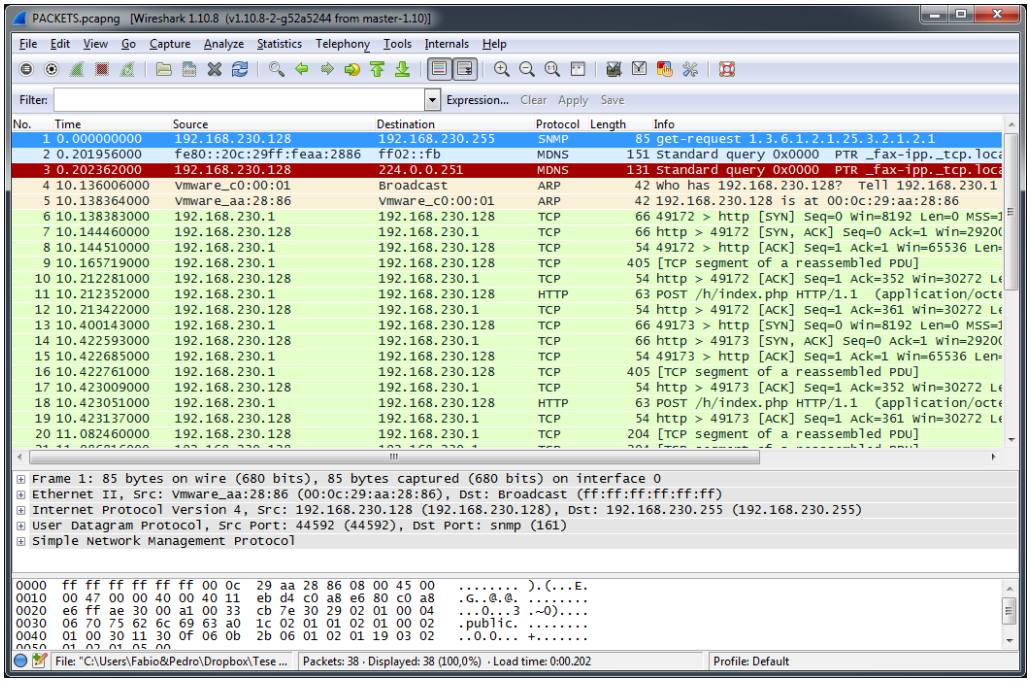


Figure 5.16: Network traffic concerning `Tinba/Zusy` communications as seen in Wireshark.

The next step is focused on the usage of ProcMon, which was started before the execution of the Malware sample in order to apply filters to all the processes running before `Tinba/Zusy` was fired up. The information gathered after the malware execution may be more easily analysed. One of the filters that can be applied is to only show actions performed by the process with the previously obtained PID or name, given the fact that we already know this information. A

representation of the malware activities obtained by the ProcMon software can be seen in figure 5.17.

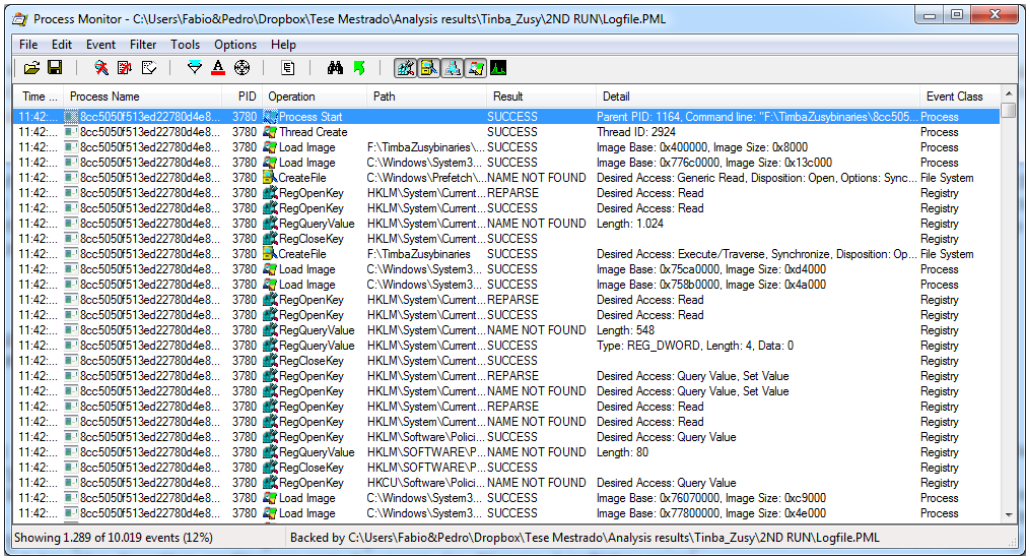


Figure 5.17: Screenshot showing the activities registered by the ProcMon utility for the `Tinba/Zusy` process.

As can be seen in the aforementioned figure, the `Tinba/Zusy` process is very active while executing. A good first step when using ProcMon is to filter all activities but the ones related with Writes. This way, it is simpler to identify files or directories created by the process. Nonetheless, one must not forget that malware often try to ask other (legitimate) processes to perform those action on its behalf. After analysing the information collected in this step, it was possible to conclude that `Explorer.exe` was probably being called by the malware sample PE, that a new directory was created on the `Program Data` folder, and that an entry was inserted in the System Registry, forcing some `bin.exe` file to execute during boot. However, that file was not in the expected folder, which means that it would probably be retrieved from the Internet if the true link was available. As the network and the php file that was requested to the server were only simulated, it is not possible to know the actions that would be taken by `Tinba/Zusy` afterwards. Figure 5.18 shows the `Windows Explorer` process creating the autorun directories and inserting the autorun entry in the System Registry.

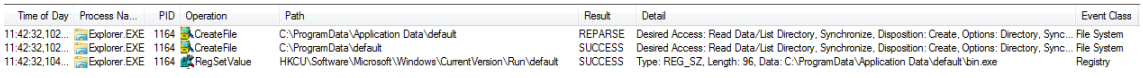


Figure 5.18: Partial screenshot showing `Explorer.exe` creating an Autorun folder and registry entry.

It should be noticed that the aforementioned AutoRun entry is only created for the current logged In user, since the key is only setted on the Handle to Registry Key for the Current User (HKCU) hive of the System Registry. These artifacts concerning the registry were also noticeable when using the AutoRuns software, of which a screenshot is included in figure 5.19.

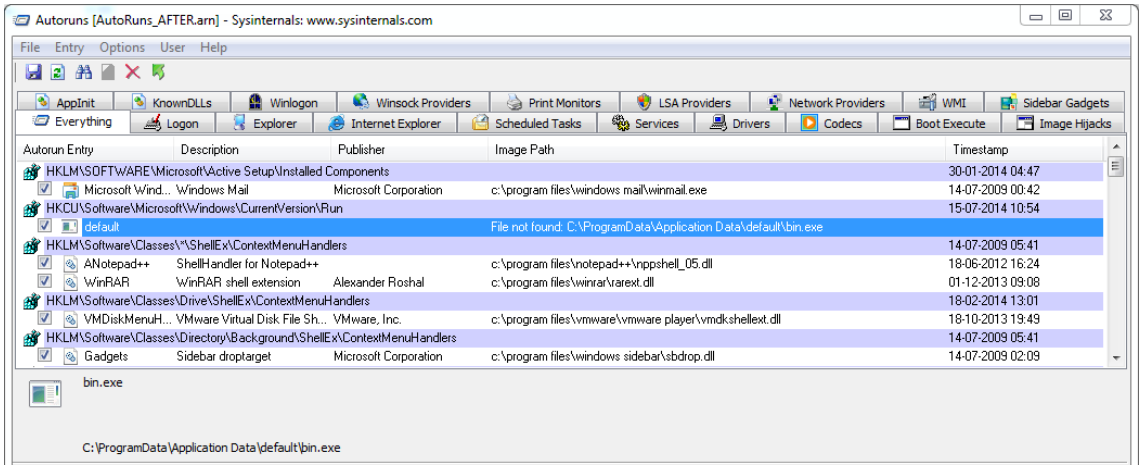


Figure 5.19: Screenshot of the AutoRuns tool showing the registry key inserted by `Tinba/Zusy`.

5.6 Conclusion

This chapter provides a brief overview over the two basic forms of malware analysis. Along with the explanation, two malware samples were used to introduce the process in practice. Regarding the `Stuxnet` samples, only very few evidences of malware were found during the basic static analysis, probably corroborating the sophistication of this well known sample. `Tinba/Zusy` was more expressive and the information gathered during the basic dynamic analysis can be summarized as follows:

- The sample communicates with the Internet, probably with the intention of downloading some files or to obtain the Command & Control Software;
- It executes a new process called `Winver.exe`;
- It creates an Autorun entry on the system registry;
- It does not replicate on its own (or, at least, its most basic form does not try to spread);
- It does not appear to possess a malicious payload, even despite leaving traces of execution on the system, which may prove to malicious later on.

The aforementioned details can now be used to guide further investigations to the malware sample. For example, it may be interesting to find out what the `Winver.exe` process was started for. On the next chapter, advanced techniques for malware analysis will be described, where this particular detail will be further analysed.

Chapter 6

Advanced Analysis

6.1 Introduction and Definition

After having discussed the results obtained using basic analysis in the previous chapter, some doubts regarding details of the `Tinba/Zusy`behaviour and purposes still persist. By now, conclusions are comprised by a set of hypothetical features and behaviours, and there is still a need to obtain more cohesive information about the malware sample. Its actions on the system, and how it accomplishes them, are to be better defined by the analysis described in this chapter. This next step is then to use advanced techniques, either static or dynamic, in order to get this information. The purpose is to simultaneously provide also some knowledge about these more advanced techniques along with the report of the empirical experiments. Notice that no advanced static analysis was attempted at this stage, mostly due to its steep learning curve. Tackling this topic would require more time. `Tinba/Zusy` will be analysed in a laboratory environment, namely in the computers mentioned in the previous chapter, which where running the Microsoft Windows OS.

This chapter is structured as follows. The next section elaborates on the differences between advanced static and dynamic analysis. The tools utilized in this phase of the work are then described in section 6.3 and the advanced analysis is discussed in section 6.4.

6.2 Advanced Static vs. Advanced Dynamic Analysis

As mentioned above, the advanced static analysis has a very steep learning curve. This translates into the requirement of spending time learning and building knowledge on low-level instructions and machine code. Apart from this effort, it is possible that a skilled analyst spends a lot of time just in this analysis phase, because the code may be very complex and long. Additionally, the results of this analysis may still be inconclusive or not as informative as the analyst thought they would be. In this section, a brief explanation for such a fact is provided, and both approaches are defined.

The advanced static analysis consists in reverse-engineering the malware compiled code and organize its internal data, which may comprise other files. To perform this task, the analyst should load the PE file into a disassembler software (some examples are included in section 6.3) and look into the instructions of the program. By investigating the instructions of the program

that are executed by the CPU at runtime, the analyst can probably find what the program does more precisely. The problem with this technique is that the analyst should be familiar with disassembly tools, as well as with code constructs (which are different for the several programming languages), and must have a deep knowledge about the OS concepts and functioning, in this case about the Microsoft Windows OSs. All this facts add up to a much more complex process than the one of advanced dynamic analysis.

Advanced dynamic analysis consists in analysing the malicious sample while it is executing, but with the help of a debugger. This technique becomes particularly useful when malware is either packed or obfuscated, since it will allow the analyst to perform its work while the code resides on the memory stack, probably unpacked and not obfuscated. Usually this technique is performed alongside with the advanced static analysis, complementing it. Cause the approach is based on the usage of debuggers and on reporting the behaviour during execution, the learning curve is not as steep as for the previously mentioned approach.

On the next section, some tools typically used by analysts to perform both of the analysis techniques will be presented. However, the focus of the explanation will be on the advanced dynamic analysis tools and techniques, and also on the findings resulting from applying them to `Tinba/Zusy`.

6.3 Techniques and Tools

Disassembly

Disassembly is most of the times associated with the *reverse engineering* term. The act of disassembling a file consists in converting the binary values, contained in the file, in machine readable language (input), into assembly language code (output). Assembly language is not actually a single language, since it varies from one microprocessor to another. Nonetheless, within the context of this Ms.C. dissertation, it is assumed that the `x86` architecture is used and, as such, the instructions that might appear in the following discussion will be referring to `x86` assembly.

Debugging

Debugging is the technique used to test or examine the execution flow of a program. This technique was designed for developers to give them a way of analysing bugs and possible mistakes occurring in the program flow. This is known as source-level debugging. However in the context of malware analysis, debuggers are not used with that objective in mind, and they may be different from the ones used in software developing. Usually, malware analysts resort to assembly-level debuggers, which are sometimes also called as low-level debuggers. They allow the analysts to check for memory locations, register and arguments of every function of the malicious samples, offering a dynamic view of the payload of a program.

Usually, two types of debugging are mentioned: *user-mode* and *kernel-mode*. The first one consists in executing the debugger on the same system as the program being debugged, since the OS is able to separate the process being debugged from the others. The second one requires the analysts to run the debugger on a different system. In this case, when executing a software in kernel-mode, the OS is not able to separate the processes, which may lead to unrecoverable errors.

In order to perform the advanced analysis on `Tinba/Zusy`, three tools were identified and explored. Below a brief description of those tools is provided:

- *IDAPro* is a disassembler entirely written in C++ language, and is the most widespread used disassembler available for malware analysis. IDAPro allows the analyst to not only perform the disassembling in multiple platforms, as also debugging, providing analysts with one of the most complete solutions towards advanced static and dynamic malware analysis. This software can be found, along with some tutorials and documentation, on its official website [SA14].

- *OllyDbg* is another useful and popular tool in this context. It provides a user-mode debugger with a GUI, which offers the analysts many options to explore in order to find useful information about the sample. An official website with documentation is also available [Yus14], and the software is free.

- *WinDbg* is a very popular debugger for Microsoft Windows OS analysts, mostly because it is freely distributed by Microsoft. It can be used to debug PE files in user- and kernel-mode. However, it requires some knowledge to be correctly handled, since it does not offer a GUI, only a CLI (which may demotivate inexperienced users) [Net14].

The next section provides a detailed analysis on the `Tinba/Zusy` malware sample, previously analysed only resorting to basic static and dynamic analysis techniques (see Chapter 5 and Appendix B). This time, some more advanced techniques and tools are going to be used.

6.4 Advanced Analysis

When arriving at the advanced analysis stage, it is important to summarize the results obtained during the previous stages, in order to understand what should be looked for next. The information gathered so far can be summarized as follows:

- the malware communicates with a remote server with the `www.dakotavolandos.com` URL,

- issues a request to a file on the remote server, named `index.php`,

- creates an autorun key on the system registry,
- copies itself into a directory,
- creates a child process named `Winver.exe`, and
- interacts with the system process called `explorer.exe`.

By looking at these findings, an analyst, should definitely be aware of a few key aspects of the execution flow. For example, one of the initial concerns should be to find out what the malware might need from the remote server (since it does not try to download any file). Other aspects include finding exactly how it creates a child process, what is that child process doing while the malware is running, and what is the `explorer.exe` process performing also. If the analyst finds information regarding these aspects then, he or she might find a few key aspects of the payload of this malware sample.

The first task was to load the PE file into IDAPro and try to obtain some information, by looking into the malware sample disassembled code. The results of this analysis were inconclusive, specially because there are too many lines of code. Due to time constraints, it was decided to try a different approach, and step into the advanced dynamic analysis immediately. As such, the next task consists in debugging the malware sample using also IDAPro, and try to analyse it while executing. However, the free version of IDAPro was not providing enough information.

Since this process is sometimes dominated by *a cat and mouse game*, the next option was to try the two aforementioned with OllyDbg instead. On OllyDbg, looking at the assembly code was not an option either, due to the reason mentioned above. The debugging process however, was more fruitful this time. The most interesting results are discussed below with the help of some figures.

After allowing the malware to execute for a certain period of time, by stepping into the instructions one by one, by means of the debugger, and by looking at the data contained on the memory stack, it was not possible to initially notice anything important. As such, a different approach was taken, consisting in searching directly for calls to specific functions that were considered important. These functions where identified with the help of the previously obtained information. Since it was known that the malware was creating a process, a call to `CreateProcessA` must be found during execution and, as such, after inserting the breakpoint on the malware at that specific point, new information could be found, as illustrated in figure 6.1.

As can be seen in the the right side of the bottom of the figure, the `winver.exe` child is actually created by the malicious process, and the creation flag value is set to `CREATE_SUSPENDED`. This is a very common technique when performing process replacement, and so it might indi-

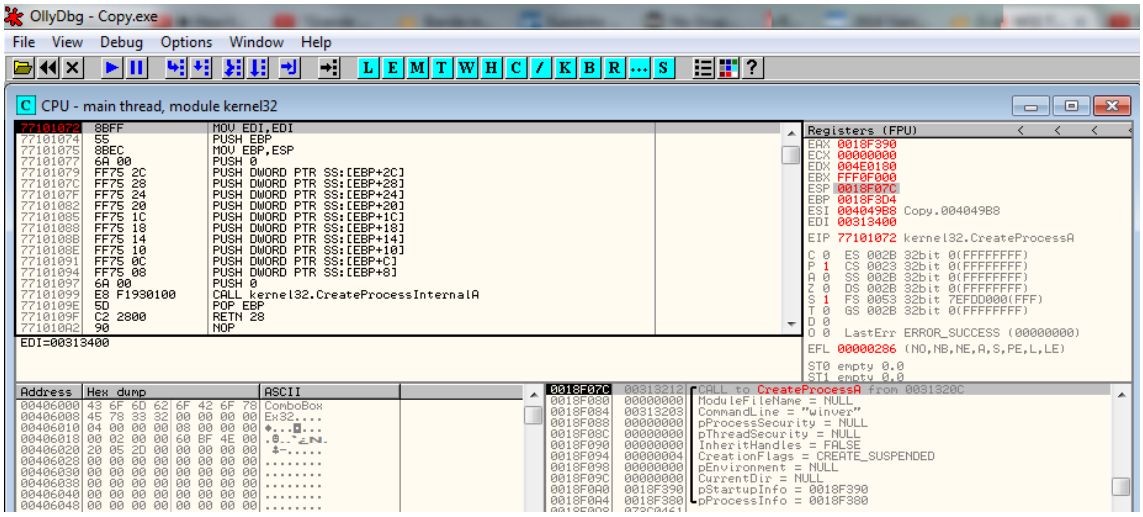


Figure 6.1: Screenshot of the OllyDbg GUI showing the call to create the `Winver.exe` process.

cate that the malware sample may try to inject a piece of malicious code into a legitimate process in the future, in order to execute it with different privileges, or even to execute it under concealment. The next strategic step was to make the execution break at the call of the method responsible for allocating resources on any running process. In Windows OSs, such call is the `VirtualAllocEx`. Figure 6.2 shows the call to this method. Though it is not shown,

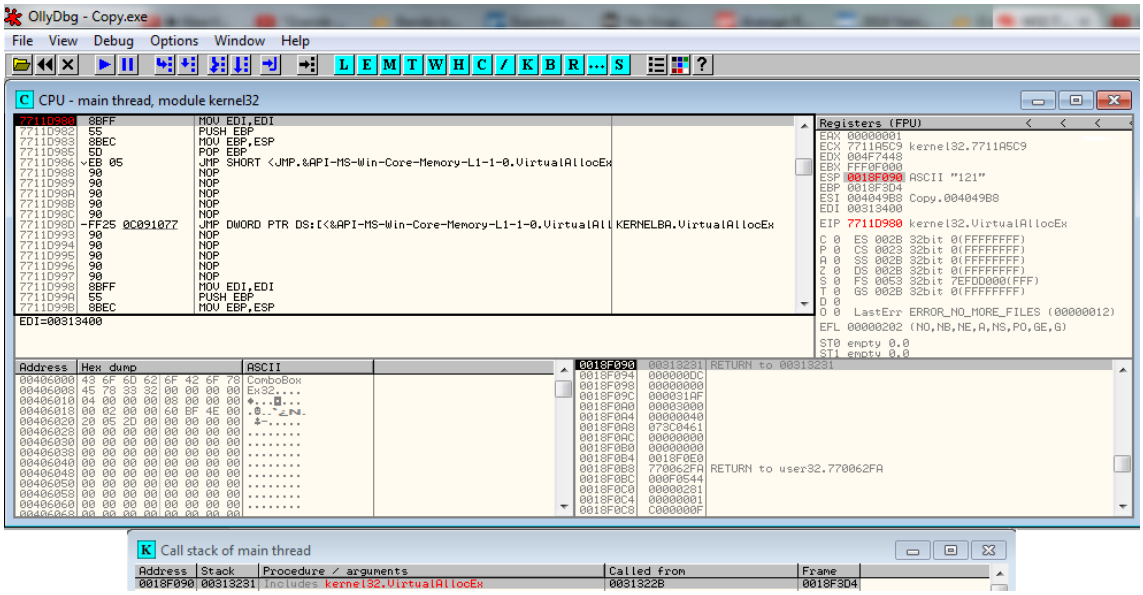


Figure 6.2: Screenshot of the OllyDbg GUI showing the call to the `VirtualAllocEx` method.

the aforementioned method also changes the permissions to allow read and write operations on that memory region. According to several sources [Mon11], it is common for sophisticated samples to later try to disguise this fact (discussed below). Still pursuing the idea that process injection would be tempted by the sample, the next step comprised a search for the call to the `WriteProcessMemory` method, typically used to achieve that objective. This call was found also, as shown in the right side of the bottom of figure 6.3. The next probable action of the malware might be to change the protection of the aforementioned region of memory, so that it looks nor-

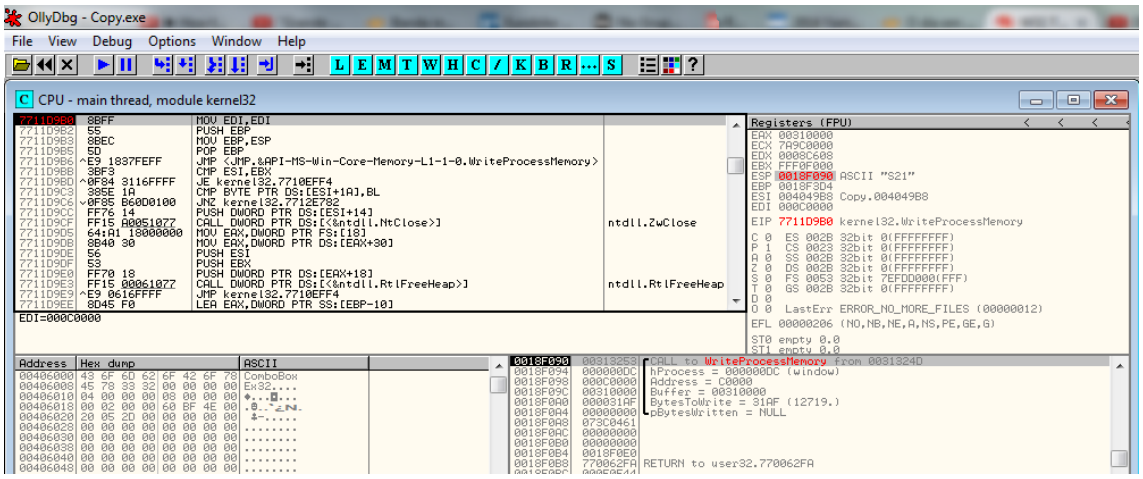


Figure 6.3: Screenshot of the OllyDbg GUI showing the call to the `WriteProcessMemory` method.

mal for Windows processes. This objective is normally achieved by using the `VirtualProtectEx` method. Thus, a breakpoint was set to where the function is called, in order to see exactly if the action is or not performed and with which parameters. This step of the analysis is depicted in figure 6.4.

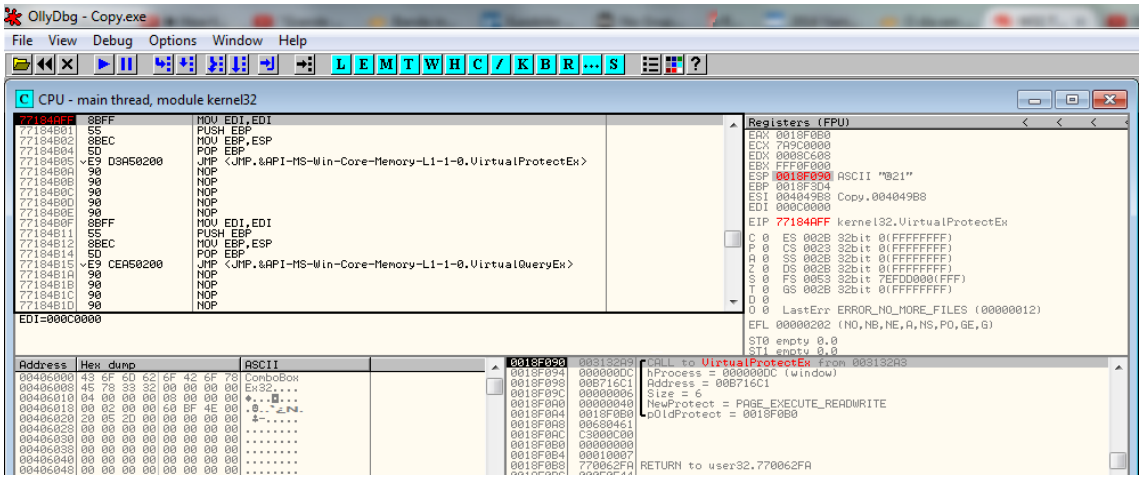


Figure 6.4: Screenshot of the OllyDbg GUI showing the call to the `VirtualProtectEx` method.

After performing all of the aforementioned tasks and system calls, the malware should try to resume the execution of the process that was initially created suspended. This is typically done using a call to the `ResumeThread` method. From this point on, it is known that the payload that was supposed to be executed by the malware is delivered via that child process. The call to this method can be seen in the *call stack* window depicted at the bottom of figure 6.5, and also at the right side of that figure, which concludes this analysis.

The analysis described in this section shows that the analysed sample in fact exhibits malicious behaviour, and it was possible to actually pinpoint some of the techniques that malware employs when infecting and hiding in a system. It was not possible to further explore the behaviour of the

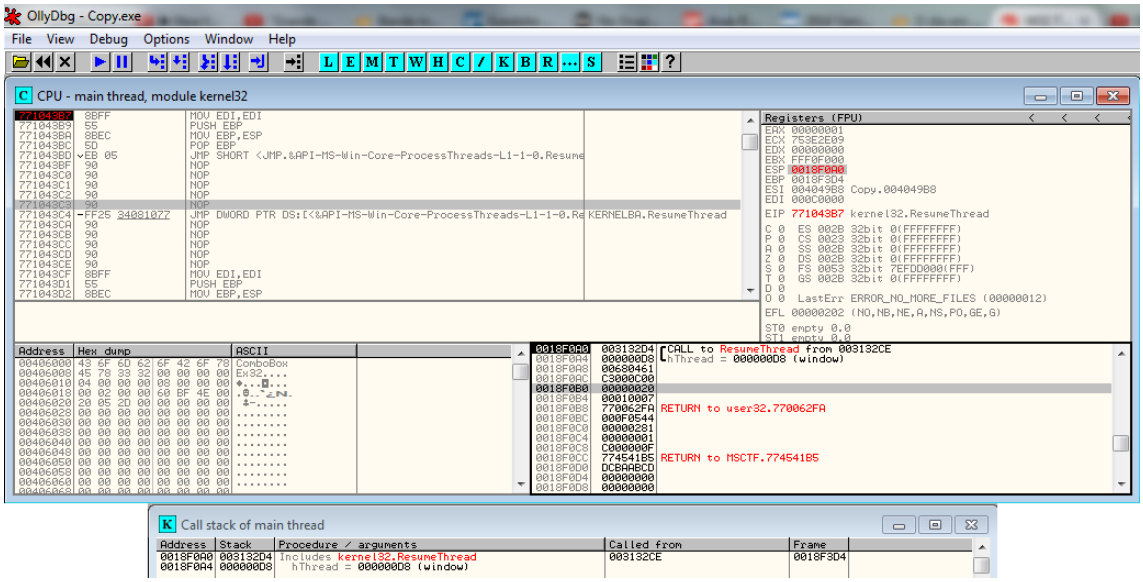


Figure 6.5: Screenshot of the OllyDbg GUI showing the call to the `ResumeThread` method.

malware, due to time limitations and to the fact that this particular sample was also somewhat dormant. Probably, it detected that the environment was not meeting the specifications.

6.5 Conclusion

This chapter elaborates on the advanced analysis approach for studying malware. Several tools that may aid specialists on this area to perform source-level debugging where mentioned and used to further investigate the `Tinba/Zusy` sample, which was already subject of analysis in the previous chapter. The advanced dynamic analysis enabled concluding that `Tinba/Zusy` implements a concealment technique known as process replacement, which is a very concrete information about this malware. It was mentioned that the advanced dynamic analysis is a simpler approach than advanced static analysis in terms of the time it takes to learn the concepts and mechanisms involved.

Interestingly, the process that `Tinba/Zusy` spawns is the `Winver.exe`, which is a legitimate Microsoft Windows utility to show information about the OS. The next stage of this analysis would be to try to obtain the code that is injected into this process, which is left as a future work guideline.

Chapter 7

Conclusions and Future Work

This chapter contains the final remarks of the work performed under this Ms.C. programme, as well as some thoughts on its subject. Some guidelines for future research and further development of this work are also pinpointed in the last section of this chapter.

7.1 Main Conclusions

The history of malware is relatively recent, when compared to the computer history and evolution. This type of software was the focus of more professional development after the worldwide adoption of the Internet, and also when it was noticed that malware could be used to gain money. The development of malware is now defined by an exponential curve and, in the last quarter of 2013, there where approximately 315000 new threats per day, according to Kaspersky Lab, which, when compared with values from 2012 (200000 threats per day), reflect the boom of malware [Kas13]. Given the proliferation of malware, which nowadays feeds an entire industry, classification systems where developed. Nonetheless, it is sometimes difficult to assign a sample malware to a single class and several malware falls into several classes. An alternative system, described at the end of chapter 2 shows the effort to produce a simplified and more general classification system, in which is easier to denote the behaviour and the targets of most of these new programs.

A malware is named after it is analysed by a computer forensic specialist. If it is part of a known malware family, it gets a new variant name, and is documented as a variant of a given family. However, if it corresponds to an unknown behaviour, a new family is defined. The specialist is the one naming his discovery. This is the main reason why malware may have different names on different anti-malware companies. If two malware analysts work on the same malware at the same time, and that malware is new, both will give it a name, making it hard to study all the variants, since they are documented differently. The lack of a standard is thus one of the main problems. Besides that, when a malware gets famous, it gets a lot of users (which use it to their own purposes) that slightly modify the program to fit their needs. This leads to the appearance of a large number of variants.

In spite of all the problems and techniques used by malware to infect machines, the human factor is perhaps, the biggest problem when it comes to malware spreading. Most of the users are not informed and trained enough to maintain their personal or corporate computers and networks

safe. Some knowledge is required to keep the systems updated and free of infections. It is crucial that the user is trained and informed about safety measures, to lower the risk inherent to the usage of Internet and Computers. Firewalls, IDSs, IPS and AV hardware or software are not 100% efficient, and as such, it becomes crucial that the users adopt a safe approach towards this situation. When it comes to corporate networks, Firewalls alone are not sufficient to keep the network safe, given the fact that they only look at incoming and outgoing traffic to the Internet, and do not monitor the inside traffic. In these situations, there is a need to implement other systems that do monitor intra-network traffic, namely IDSs or IPSs. A 100% accurate defence system does not exist, and it should be tailored to the specific scenario so as to be as effective as possible.

The more recent Cloud-based AVs have their advantages, namely the low processing burden on the computer of the user, the live updated and typically larger database and the fact that they are free or cheap. However, they have some disadvantages too, namely when it comes to the privacy and confidentiality of the documents of the user. From the discussion on AVs, it can be concluded that traditional solutions are still seen as more reliable comparatively to these new solutions.

During this research programme, it was noticed that most of the malware families and variants found worldwide might present a big threat to computer systems, specially when they contain some malware hiding or data encoding mechanisms. These features may circumvent heuristic algorithms, and might trick the system into allowing a malicious program in. Thus, it is of utmost importance to have the system updated in order to try to diminish the possibilities of such an outbreak. A malware family tree is included in Chapter 3 which, in the opinion of the author and even though it mentions only 24 families in a vast panoply of malware, it provides a good perspective over the heterogeneity that dominates this area. This graphical representation, along with the documentation presented herein, constitutes a good starting point for practitioners entering this area.

The first real difficulty was faced after performing the revision of the state of the art. It was hard to find a malware sample that could be used for analysis purposes. As malware can be categorized differently, according to the different companies, and may have various different variants, it is hard to find a malware variant which might be of interest to analyse. This was a challenge, specially due to the mysticism underlying this topic. Malware samples are sometimes released to the public, via proper channels or dedicated blogs. Most of the times, they are sold and are very difficult to obtain, unless recurring to underground communities.

Several problems were faced during the analysis of malware samples, namely in what refers to the available documentation. It was needed to gather information from both electronic and paper sources. Nonetheless, the information was not sometimes coherent between them. As this

is a vast area, dominated mostly by the specialized industry, some of the information collected may still not be 100% accurate. There was a significant effort in making all the information contained herein coherent.

As mentioned, it is hard to find a particularly good malware sample online, specially because most of the malware known worldwide may explore a particular flaw that has already been corrected by the responsible parties. On top of that, the malware used in the scope of this Ms.C. programme might not behave the same way as it would in the environment for which it was released, meaning that, normally one only has access to the samples some time after it was released, or fulfilled its purpose. Recreating all the variables and the environment of a given sample is difficult. This added some unpredictability to the analysis performed herein. There is still a lot of work that needs to be done in order to ease the aforementioned task.

During this Ms.C. programme, it was possible to analyse two malware samples. To the best of the knowledge of the author, Chapter 5 and 6, along with Appendix B, comprise the most detailed documentation available for the `Tinba/Zusy` malware sample. The analysis corroborates that the malware samples were the brilliantly engineered worm called Stuxnet and the `Trojan-Banker Tinba/Zusy`. During the several phases of the analysis, it was possible to emphasize some of the behaviours that could indicate their specific category.

Even though it was not possible to observe all the features that define Stuxnet, probably due to the sample that was found and used, or due to the OS configurations, it could be concluded that the program was not packed, and that it had dynamically linked libraries, meaning that it was trying to execute unnoticed in the system. This was observed using basic static analysis. It was also noticed that the program tried to delete files from the system and also that this worm was probably not able to deliver its full payload (the samples were probably simplified or dormant versions of the worm). Because of that, a different malware sample was then selected: `Tinba/Zusy` malware.

The basic static analysis to `Tinba/Zusy` showed that the program was probably performing process or DLL injection, contained a GUI and was probably not packed. This information drove the remaining part of the analysis. The basic dynamic analysis enabled the investigation of communications with network resources, leaving the Virus category out of scope. The sample was executing a new process, called `winver.exe`, and it does not replicate on its own, which places it in the trojan category. Further analysis enabled the exclusion of subclasses as the Trojan-FakeAV, Trojan-ArcBomb, Trojan-SMS, Trojan-Ransom, etc., leaving the Trojan-Banker as one of the most probable options.

In conclusion, when analysing malware samples, the analyst should always keep an open mind on what he or she might face. When handling an apparently complex malware, the analyst

should consider applying a panoply of techniques as wide as possible, so as to gather the highest amount of information possible before jumping into conclusions. Just like detection systems, no analysis technique or tool is flawless nor complete, and if the malware developer knows them he or she can always end up using techniques for circumventing or fooling them.

As shown in chapter 5, a basic static analysis, although useful as a first approach towards malware analysis, can not be used to obtain information about the full behaviour and payload of a malware sample. Nonetheless, the information gathered using this approach is sufficient to identify a few possible artefacts and behaviour, which might be used to adapt the remaining line of analysis and fine tune a safe-environment to a further advanced analysis phase. During the basic static analysis, one of the most important steps is, in the opinion of the author, to submit the file into any available online sandbox. This can provide one with the feedback that can drive the analysis further. If the sample submitted to the online sandbox is detected as malicious by any of the available anti-malware solutions, then it is immediately tagged as malicious. If duly categorized, then some of the expected behaviours are already known. On the other hand, if the results provided by the online sandbox do not present any conclusive remark, it is of utmost importance to inspect the results of the remaining techniques and tools with greater detail. Every detail found in this phase may be crucial in advanced phases of the analysis.

The next most important step is performing basic dynamic analysis. Some of the most important details gathered in this phase are related with the files and registry keys that the threat creates. From this information, it is possible to build anti-malware rules focused on the specific sample. Also in this phase, it is possible to identify all the processes created during the execution of the sample. The network communications are thus an important aspect in this part of the work, since malware often communicates with network resources in order to propagate, request information, or simply download files to the infected machine. Any anti-malware system might opt to filter network communications of those processes, opening the possibility to detect possible malicious software simply by observing the network activity. One should keep in mind that this phase relies on the correct execution of the malware samples, which may not always be possible. Therefore, and even though it is cumbersome and time consuming, the advanced analysis might prove useful when analysing a malware sample. When analysing malware, the PE file and other resources are usually available, but not to its source code. It is possible to develop anti-malware rules based on disassembled code or in specific calls within that code. Although it is necessary to apply a significant effort in this task, it is perhaps where the biggest amount of information can be retrieved. By unifying all the artefacts and rules into a single or composed rule, one may obtain a strong detection mechanism, for a specific sample.

7.2 Future Work

As this is such a vast and somehow unexplored area, it is difficult to enumerate all possible research directions related with the tasks at hands. Below are listed and described a few of them which most directly derive from this work.

Improve Assembly language knowledge - One of the aspects that can positively affect the process of malware analysis is the improvement of the means to visualize and understand assembly language. Without the proper assembly language knowledge, which can be time consuming, it may become hard to analyse any malware sample, since they are typically very complex.

Research on Packing and Obfuscation - Packing and Obfuscation reversing deserve to be more explored, focusing on, perhaps, producing a more unified approach to this matter. This requires studying all the possible packing and obfuscation algorithms, all the techniques that an analyst might use in order to reverse them, and all anti-debugging and anti-disassembly techniques, searching for ways to improve the associated tools and ultimately devise methods to overcome them.

Android OS and other OSs malware analysis - As mentioned in section 1.1, the Android OS is one of the most popular mobile platforms in terms of malware. The analysis of malware for this system is a very prominent research topic.

Automatic Crawler software - Most of the strings found during the analysis of a PE file concern a Microsoft Windows API or any method contained in one of them. Since the analyst might not possibly know every single method and API from Windows, an automatic search engine to perform this task would be useful. The main objectives of this crawler are to perform an automatic online search in a few hardcoded website URLs, to obtain some information about the windows APIs and methods, which can then be identified when analysing the strings of the PE file. This crawler could even have a generalization module, based on a database, which then could deliver the analyst the most probable behaviours of the specific malware sample (s)he is analysing, and the possibility of saving the searched strings in memory to speed up a future search.

Information and file sharing platform - There are already some communities and forums devoted to sharing information regarding malware. Nonetheless, as stressed out in this dissertation, it is not typically done in a cohesive and coherent manner, making the job of new researchers specially difficult. The creation of an information and file sharing platform, with support of the anti-malware companies, might help dispersing some of the mysticism surrounding this area. With this platform, the industry and academia could provide better and more accurate studies on this topic.

Repository of deactivated malware samples - The development of a centralized repository of deactivated malware samples would also be a potential interesting line of work. This repository would have to contain a mechanism for guaranteeing that the shared malware samples were no longer active, since it would open the world of malware to more users. This repository would perhaps help researchers to improve their knowledge on the subject.

Basic Analysis tool - Since basic static analysis is based on existing software, it makes sense to make an effort to fully automate this task. This could perhaps be made using by developing a standalone and free tool, with which a larger number of analysts could contribute to the analysis of samples, while focusing on more advanced topics. Nowadays, there are some automated tools for performing such task, however, they are paid or web-based, meaning that the samples have to be sent to a remote location for analysis. The tool would enable local analysis and could be used in situations were private data was involved.

GUI implementation for some of the tools - Several different software solutions were used within the scope of this Ms.C. programme. Some of those tools do not present a clean and useful GUI and, in some cases, it would make sense to implement one. For example, it could ease the configuration process. Some tools which would benefit from this improvement are *InetSim*, *Strings* and *WinDbg* tool.

Analysis of malware samples - One of the most direct lines of work consists of keeping analysing new malware samples, or revisiting the ones explored along this work with different approaches and tools. A possible and interesting sample that may be analysed next is the `Zeus/Zbot trojan`.

Bibliography

[Acu12] Acunetix. What is Malware. http://www.websitedefender.com/what-is-malware/, 2012. Last access: February 05, 2014. 9

[Bon] Dr. Vesselin Bontchev. Current Status of the CARO Malware Naming Scheme. http://www.people.frisk-software.com/~bontchev/papers/naming.html. Last access: February 05, 2014. 31

[CAR02] CARO. CARO Naming Scheme. http://www.caro.org/naming/scheme.html, 2002. Last access: February 05, 2014. 31

[Cen14] Microsoft Malware Protection Center. Naming malware. http://www.microsoft.com/security/portal/mmpc/shared/malwarenaming.aspx, 2014. Last access: February 05, 2014. 31

[CG13] Christian Chebyshev, Victor;Funk and Maria Garnaeva. IT Threat Evolution on Q3 of 2013 from Kaspersky. http://www.securelist.com/en/analysis/204792312/IT_Threat_Evolution_Q3_2013, November 2013. Last access: February 05, 2014. 35

[CGN] Alessandro Tanasi Claudio Guarnieri and Andy Nordbo. Malwr - free malware analysis service and community. https://malwr.com/. Last access: August 21, 2014. 65

[Cir08] All About Firewalls. http://firewall-review.narod.ru/circuit_level_gateway.html, August 2008. Last access: February 05, 2014. 55

[CmR04] Thomas M. Chen and Jean marc Robert. The Evolution of Viruses and Worms. In *Statistical Methods in Computer*, 2004. 10

[Com10] CompareBusinessProducts.com. IDS vs. IPS Explained. http://www.comparebusinessproducts.com/fyi/ids-vs-ips, March 2010. Last access: February 05, 2014. 57

[Cor14a] Dice Holdings Corporation. Regshot Download page. http://sourceforge.net/projects/regshot/, 2014. Last access: August 21, 2014. 66

[Cor14b] Faronics Corporation. Data Igloo. http://www.faronics.com/en-uk/products/data-igloo/, 2014. Last access: July 15, 2014. 75

[Cor14c] Faronics Corporation. Deep Freeze Software Features. http://www.faronics.com/en-uk/products/deep-freeze/enterprise/, 2014. Last access: July 11, 2014. 75

[CPST14] Inc. Check Point Software Technologies. Firewall Overview. http://www.checkpoint.com/resources/firewall/, 2014. Last access: February 05, 2014. 55

[Cra] Stephanie Crawford. How a Cloud Antivirus Works. http://computer.howstuffworks.com/cloud-computing/cloud-antivirus.htm. Last access: February 05, 2014. 60

[Dow07] Mike Dowler. Circuit Level Gateways. http://www.pcstats.com/articleview.cfm?articleid=1450&page=5, July 2007. Last access: February 05, 2014. 55

[DS13] Dr.(Mrs).G.Padmavathi and S.Divya. A Survey on Various Security Threats and Classification of Malware Attacks, Vulnerabilities and Detection Techniques. http://www.journalofcomputerscience.com/2013Issue/Jun13/V2No04Jun13P006.pdf, 2013. Last access: March 20, 2014. 12

[ESE13] ESET. October 2013 Threat Report. http://www.virusradar.com/sites/default/files/reports/Threat Radar Report October 2013.pdf, 2013. Last access: February 05, 2014. 34

[ESE14] ESET. ESET Statistical top 10 actual threats per country. http://www.virusradar.com/en/statistics/10, 2014. Last access: February 05, 2014. 36

[Fou14a] Cuckoo Foundation. Cuckoo Sandbox Malware Analysis System website. http://www.cuckoosandbox.org/, 2014. Last access: August 21, 2014. 65

[Fou14b] Wireshark Foundation. What's on your network?, wireshark website. https://www.wireshark.org/, 2014. Last access: August 21, 2014. 67

[GKAB09] A. Gupta, P. Kuppili, A. Akella, and P. Barford. An empirical study of malware evolution. In *Communication Systems and Networks and Workshops, 2009. COMSNETS 2009. First International*, pages 1--10, Jan 2009. 9

[HE14a] Thomas Hungenberg and Matthias Eckert. INetSim: Internet Services Simulation Suite configuration information webpage. http://www.inetsim.org/documentation.html, 2014. Last access: June 19, 2014. 75

[HE14b] Thomas Hungenberg and Matthias Eckert. INetSim: Internet Services Simulation Suite webpage. http://www.inetsim.org/, May 2014. Last access: August 21, 2014. 67

[Hyp11] Mikko Hypponen. Defcon : The History and evolution of malware. http://www.youtube.com/watch?v=L8lA1pNvcz4, January 2011. Last access: February 05, 2014. 1, 4

[ICP] Stavros D. Nikolopoulos Ioannis Chionis and Iosif Polenakis. A Survey on Algorithmic Techniques for Malware Detection. http://cs.uoi.gr/~ichionis/ISCIM'13-CNP-32.pdf. Last access: March 20, 2014. 13

[ICS11] Internet-Computer-Security.com. IPS and IDS. http://www.internet-computer-security.com/Firewall/IPS.html, 2011. Last access: February 05, 2014. 57

[Ina14] Pedro Inacio. Lecture notes in computer security. Universidade da Beira Interior, Department of Computer Science, 2014. 55, 57

[Joh11] Angus Johnson. Resource Hacker Software download page. http://www.angusj.com/resourcehacker/, September 2011. Last access: March 28, 2014. 114

[Kas10] Michael Kassner. How Antivirus Software works. http://www.techrepublic.com/blog/it-security/how-antivirus-software-works-is-it-worth-it/#., January 2010. Last access: February 05, 2014. 57, 58

[Kas13] Kaspersky. Number of the Year. http://www.kaspersky.com/about/news/virus/2013/number-of-the-year, December 2013. Last access: February 05, 2014. ix, 7, 31, 91

[KM13] Saroj Hiranwal Kirti Mathur. A Survey on Techniques in Detection and Analyzing Malware Executables. http://www.ijarcsse.com/docs/papers/Volume_3/4_April2013/V3I4-0290.pdf, 2013. Last access: March 20, 2014. 11

[Lab13a] Antiy Labs. Hacktool Families. http://www.virusview.net/description/classification/hacktool, 2013. Last access: February 05, 2014. 37

[Lab13b] Antiy Labs. Trojan Families. http://www.virusview.net/description/classification/trojan, 2013. Last access: February 05, 2014. 37

[Lab13c] Antiy Labs. Virus Families. http://www.virusview.net/description/classification/virus, 2013. Last access: February 05, 2014. 37

[Lab13d] Antiy Labs. Worm Families. http://www.virusview.net/description/classification/worm, 2013. Last access: February 05, 2014. 37

[Lav13] Lavasoft. History of Malware. http://www.lavasoft.com/mylavasoft/company/blog/history-of-malware, November 2013. Last access: February 05, 2014. 1, 4

[Lie12] Matt Liebowitz. Microsoft mistakenly flags Google as malicious site. http://www.nbcnews.com/id/46404138/ns/technology_and_science-security/t/microsoft-mistakenly-flags-google-malicious-site/#.VC0WJBZEPqc, February 2012. Last access: September 20, 2014. 59

[Man14] A FireEye Company Mandiant. Mandiant ApateDNS - Control DNS Responses software information webpage. https://www.mandiant.com/resources/download/research-tool-mandiant-apatedns, 2014. Last access: August 21, 2014. 66

[Mic] Microsoft. The evolution of malware and the threat landscape – a 10-year review. 1

[Mic13] Microsoft. Microsoft Security Intelligence Report. http://download.microsoft.com/download/5/0/3/50310CCE-8AF5-4FB4-83E2-03F1DA92F33C/Microsoft_Security_Intelligence_Report_Volume_15_English.pdf, 2013. Last access: February 05, 2014. 32

[Mic14] Microsoft. Microsoft Security Portal. http://www.microsoft.com/security/portal/mmpc/default.aspx, 2014. Last access: February 05, 2014. 37

[Mil06] Steve P. Miller. Dependency Walker Software download page. http://www.dependencywalker.com/, 2006. Last access: March 28, 2014. 64, 114

[Min] Jiang Ming. A Survey of Malware Analysis Techniques. http://www.personal.psu.edu/jum310/blogs/jiang_ming/2013/04/a-survey-of-malware-analysis-techniques-poster-of-ist-501.html. Last access: March 20, 2014. 10, 11

[Mon11] Eric Monti. Analyzing Malware Hollow Processes. http://blog.spiderlabs.com/2011/05/analyzing-malware-hollow-processes.html, May 2011. Last access: October 02, 2014. 87

[Net14] Microsoft Development Network. WDK and WinDbg download webpage. http://msdn.microsoft.com/en-us/windows/hardware/hh852365.aspx, 2014. Last access: August 28, 2014. 85

[New09] *Computer Security: Protecting Digital Resources*, chapter 10, pages 273--274. Jones and Bartlett Publishers, Inc., USA, 1st edition, 2009. 57

[Pas11] Paolo Passeri. Application Firewall. http://hackmageddon.com/2011/10/07/next-generation-firewalls-and-web-applications-firewall-qa/, October 2011. Last access: February 05, 2014. 55

[PEV14] PEView Software download page. http://wjradburn.com/software/, September 2014. Last access: March 28, 2014. 65, 114

[PV] Vinod P. and M.S.Gaur V.Laxmi. Survey on Malware Detection Methods. http://www.security.iitk.ac.in/contents/events/workshops/iitkhack09/papers/vinod.pdf. Last access: March 20, 2014. 12

[RC14a] Mark Russinovich and Bryce Cogswell. Windows Sysinternals Autoruns for Windows download page. http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx, 2014. Last access: August 21, 2014. 66

[RC14b] Mark Russinovich and Bryce Cogswell. Windows Sysinternals Process Monitor download page. http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx, 2014. Last access: August 21, 2014. 65

[RRC11] Dr. G.C. Hazarika Rizwan Rehman and Gunadeep Chetia. MALWARE THREATS AND MITIGATION STRATEGIES: A SURVEY. http://www.jatit.org/volumes/research-papers/Vol29No2/3Vol29No2.pdf, 2011. Last access: March 20, 2014. 13

[Rus13] Mark Russinovich. Windows Sysinternals Strings v2.52. http://technet.microsoft.com/en-us/sysinternals/bb897439, 2013. Last access: March 25, 2014. 64, 112

[Rus14] Mark Russinovich. Windows Sysinternals Process Explorer download page. http://technet.microsoft.com/pt-pt/sysinternals/bb896653.aspx, 2014. Last access: August 21, 2014. 66

[SA14] Hex-Rays SA. IDAPro Official Website. http://www.hex-rays.com, September 2014. Last access: August 28, 2014. 85

[Sec11] Panda Security. Technology Highlights: antivirus, anti-malware, Cloud ... Collective Inteligence. http://www.pandasecurity.com/about/panda-technologies/, 2011. Last access: February 05, 2014. 60

[SH12] Michael Sikorski and Andrew Honig. *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, San Francisco, CA, USA, 1st edition, 2012. 60, 64, 67

[Sha13] Vivek Shah. A Survey on Malware Analysis Techniques. http://www.researchgate.net/publication/236827365_A_Survey_on_Malware_Analysis_Techniques, 2013. Last access: March 20, 2014. 11

[SM] Karen Scarfone and Peter Mell. Guide to Intrusion Detection and Prevention Sys-

tems. http://csrc.nist.gov/publications/nistpubs/800-94/SP800-94.pdf. Last access: February 05, 2014. 57

[Sna08] Snaker. PEiD software download webpage. http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml, 2008. Last access: March 26, 2014. 64, 113

[SSA13] Imtithal A. Saeed, Ali Selamat, and Ali M. A. Abuagoub. Article: A Survey on Malware and Malware Detection Systems. *International Journal of Computer Applications*, 67(16):25--31, April 2013. Published by Foundation of Computer Science, New York, USA. 12

[Tay14] Thomas Taylor. What is Heuristic Antivirus Detection. http://internet-security-suite-review.toptenreviews.com/premium-security-suites/what-is-heuristic-antivirus-detection-.html, 2014. Last access: February 05, 2014. 59

[tea] Virus Total team. Virus Total malware analysis service. https://www.virustotal.com. Last access: March 25, 2014. 63, 64, 111

[Wik14a] Wikihow. How to Avoid Getting a Computer Virus or Worm. http://www.wikihow.com/Avoid-Getting-a-Computer-Virus-or-Worm, September 2014. Last access: October 02, 2014. 51

[Wik14b] Wikipedia. Application Firewall. http://en.wikipedia.org/wiki/Application_firewall, February 2014. Last access: February 05, 2014. 55

[Wik14c] Wikipedia. Malware. http://en.wikipedia.org/wiki/Malware, September 2014. Last access: February 05, 2014. 9

[Win] WinMD5.com. WinMD5 software webpage. http://www.winmd5.com/. Last access: March 25, 2014. 64, 112

[WN07] Brian Witten and Carey Nachenberg. Malware Evolution: A Snapshot of Threats and Countermeasures in 2005. In Mihai Christodorescu, Somesh Jha, Douglas Maughan, Dawn Song, and Cliff Wang, editors, *Malware Detection*, volume 27 of *Advances in Information Security*, pages 3--15. Springer, 2007. Available from: http://dblp.uni-trier.de/db/series/ais/ais27.html#WittenN07. 10

[Yus14] Oleh Yuschuk. OllyDbg official website. http://www.ollydbg.de, February 2014. 85

[ZAO] Kaspersky Lab ZAO. Viruses and Worms. http://www.securelist.com/en/threats/

detect/viruses-and-worms, December. 20

[ZAO14a] Kaspersky Lab ZAO. Adware, Pornware and Riskware. http://securelist.com/threats/adware-pornware-and-riskware/, 2014. Last access: February 05, 2014. 22

[ZAO14b] Kaspersky Lab ZAO. An alternative approach to classifying detected objects. http://securelist.com/threats/alternative-classifications/, 2014. Last access: February 05, 2014. 26

[ZAO14c] Kaspersky Lab ZAO. Malicious Programs. http://www.securelist.com/en/threats/detect/malicious-tools, December 2014. Last access: February 05, 2014. 13

[ZAO14d] Kaspersky Lab ZAO. Malware. http://www.securelist.com/en/threats/detect/malware, December 2014. Last access: February 05, 2014. 16

[ZAO14e] Kaspersky Lab ZAO. Malware Classifications. http://www.kaspersky.com/internet-security-center/threats/malware-classifications, 2014. 9

[ZAO14f] Kaspersky Lab ZAO. Rules for Naming Detected Objects. http://securelist.com/threats/rules-for-naming/, 2014. Last access: September 02, 2014. 31

[ZAO14g] Kaspersky Lab ZAO. Suspicious Packers. http://www.securelist.com/en/threats/detect/suspicious-packers, 2014. Last access: February 05, 2014. 22

[ZAO14h] Kaspersky Lab ZAO. The classification tree. http://securelist.com/threats/the-classification-tree/, 2014. Last access: February 05, 2014. 9

[ZAO14i] Kaspersky Lab ZAO. Trojan Programs. http://securelist.com/threats/trojans/, December 2014. Last access: September 02, 2014. 16

Appendix A

Useful Definitions, Library Linking and Windows Registry

A.1 Packing and Obfuscation

It is important for malware developers to find new and more effective solutions to hide the malicious activities of their programs making them hard to be detected or analysed. Obfuscation is the technique that mostly contributes to turning static analysis a complicated and highly unsuccessful task. This technique consists on the application of an obfuscation algorithm, that changes and makes the code harder to understand and interpret by an human.

As for a packed program, is a particular case of Obfuscation where the entire program is compressed and impossible to analyse statically. Program Packing is achieved by packing the entire program, creating a small specific program that has the responsibility of unpacking the file at runtime. On the context of malware analysis, only the small program can be analysed, thus preventing the analysis of the correct executable.

On both cases, in order to obtain conclusive information, the reverse engineering of the code to perform the static analysis of the file is a complicated task that may not give much information to the analyst.

Some ways to detect if a file is either packed or obfuscated is by looking at the imported functions. If the file has few imports and has the `LoadLibrary` and `GetProcAddress` imports, it is usually an indicator that the file is packed. Also, a low number of strings may be a proof of a packed executable. Another indicator may be the `.text` section on the PE file header, that has a size of raw data and virtual size not similar.

A.2 Library Linking

The best information which can be gathered from a PE file on the basic static analysis stage is the list of functions that it imports. The functions used by a program that are present on another program or library are called *Imports*. Those libraries can then be linked to the main executable by linking. Programmers, usually use libraries and functions in order to prevent the replication of code, since most of the times, it is already implemented in some existing library. Knowing the linking method is crucial in malware analysis, since the PE file header changes according to the linking method. Linking may occur in one of three ways, either statically, at runtime or

dynamically.

Static linking is the less used one, since it copies all the library code to the executable, making the file bigger. The size of the executable is not the only problem associated with this. When analysing executables with this linking method, it becomes hard to distinguish between the executable code and the linked functions. Nonetheless, this method is still very popular among UNIX and Linux OSs.

As for *runtime linking*, is the most popular library linking method among malware developers. However, it is when malware is either packed or obfuscated that this gets important, since, it proves as a special utility to malware developers by performing the library linking when the malware function needs it. This way, it turns difficult to mention which functions or libraries the malware uses. `LoadLibrary` and `GetProcAddress` are the most commonly used libraries on Windows OSs to import functions not present on the PE header. These imports allow a program to access any function present in any of the libraries of the system. Due to this fact, the malware analyst cannot statically claim which functions are going to be called by the malware, reason why, malware developers give this linking method priority.

Even despite the runtime linking popularity, *dynamic linking* is, nowadays, the mostly used linking method. Dynamic linking makes the host OS search for the needed libraries and functions, however only allowing for the execution of the required function when called by the program. The function is then executed inside the library file, without copying itself to the malware executable. The PE header when this method is used, contains the entire information about the functions that are going to be called and utilized by the malware program.

Identifying the libraries needed by the malware is an important source of information to statically understand what the program will do when executed.

A.3 Portable Executable File Format

On this section, the PE file format is explored. The PE file format is the one that is used to provide information to the Windows OS loader to execute the program. PE files are usually subdivided into 4 sections, which are described as follows:

1. The `.text` section is the one that contains the CPU executed instructions. Usually, this is the only section on the PE file that is able to include, and execute, code;

2. The `.rdata`, is the section that stores information about the imports and exports. The data under this section is usually the information obtained when resorting to programs like *Dependency Walker* and *PEview*;

3. As for the `.data` section usually stores the global data of the program, which can be fully accessed from anywhere in the program; and

4. The `.rsrc` section, which usually is the section where icons, images and other resources are included.

However, these 4 sections may be named differently by different compilers, and may not be the only sections of a PE file header. On top of the 4 described sections, there are some more, which can be classified as:

1. .idata - This section is not usually present on the PE file, however when present keeps the imported functions information. When this section is not present, its information is stored on the `.rdata` section.

2. edata - This section, when present is responsible to save the information about the exported functions. When not present the exported functions information is stored on the `.rdata` section.

3. .pdata - This section is only present on 64-bit executables. It is responsible to keep the information about handling the executables exceptions.

4. .reloc - It contains information that is used if the PE file needs to relocate library files.

A.4 Strings

In computer programming, a string is a sequence of characters, which forms a human readable information. Strings might store messages that should be printed in case of the occurrence of any error, URLs to any necessary online resource, and much more information.

Strings might be stored on disc in both Unicode and ASCII format. These formats define the way that the string is encoded. In ASCII encoding, the strings are only allowed to use 1 byte per character, which translates in the possibility of storing only 128 different characters, while on the most used standard for Unicode strings, the `UTF-8` standard, the strings use 1 byte in all 128 ASCII characters and up to 4 bytes in other characters. However, `UTF-8` is not capable of representing all the existing Unicode characters, which are more than 110000. To be able to represent them all, is used the `UTF-16` standard, which is an extension of `UCS-2` standard (which is now obsolete), which used two bytes for every character, however it could not encode, as well, all of nowadays existing Unicode characters. `UTF-16` represents all the characters that `UCS-2` was able to represent, using 2 bits, and adds another 4 bits to represent all of the other characters that `UCS-2` could not. Most operating systems nowadays use the Unicode encoding, and so, since this report focus on malware, and on Windows, it is out of curiosity important to

mention that every string on this OS is an Unicode string.

Strings in both encoding standards have one thing in common, namely a NULL terminator character.

All the strings after their corresponding sequence of characters, have another character encoded like *0x00* that is the
0 character which indicates that the string is over, and the characters to extract are the ones before this character.

However, when retrieving strings from any type of file (not only from PE files), not all of the byte sequences present represent strings, they can be CPU instructions, data used by the program, memory addresses but they still can have the NULL terminator character on them, and due to that trick the string retrieving software into consider it a string. It is due to this that many of the extracted strings of any kind of file might not be real strings, and might contain no useful information when it comes to human reading.

A.5 Microsoft Windows OS System Registry

The system registry exists in many OSs however, for the purpose of this Ms.C. dissertation, every time we refer to the system registry we pretend to refer to the Microsoft Windows OS system registry. This section, wishes to give a brief presentation of the Microsoft Windows OS system registry.

The Windows registry as it is called the system registry of the Microsoft Windows OS, is a central database for the Windows OS. It is responsible for keeping all the settings used to control hardware configuration, OS configuration, installed software configurations and user preferences. If any change is performed on any software, it most certainly is saved in the registry. The registry usually has thousands of entries and they are all organized according to five top-level hives presented next:

- HKEY_CLASSES_ROOT - Usually stores information about the registered programs and the file associations.
- HKEY_CURRENT_USER - Stores the settings that are specific to the current user. This is where the multiple user accounts settings are kept.
- HKEY_LOCAL_MACHINE - Responsible to keep the settings for all the users of the computer.
- HKEY_USERS - Stores sub-keys that correspond to the HKEY_CURRENT_USER keys for each user account.

- HKEY_CURRENT_CONFIG - Stores the keys generated during boot. This section is created when the computer boots and is not stored on the hard drive.

The Windows OS stores the registry recurring to a binary file format. Those files and their association to the top-level hives are as follows:

- Sam - Associated to the HKEY_LOCAL_MACHINE\SAM key.
- Security - Associated to the HKEY_LOCAL_MACHINE\SECURITY key.
- Software - Associated to the HKEY_LOCAL_MACHINE\SOFTWARE key.
- System - Associated to the HKEY_LOCAL_MACHINE\SYSTEM key.
- Default - Associated to the HKEY_USERS\DEFAULT key.
- Ntuser.dat - Associated to the HKEY_CURRENT_USER hive.

The files mentioned before, can be found in two different directories in Microsoft Windows 7 OS. The first five files can be found in C:\System32\Config\... and Ntuser.dat can be found in %USERPROFILE%\Ntuser.dat. This is meant to provide the system the ability of loading global and individual configurations upon startup and login, and improving user abstraction.

When looking into the registry on regedit.exe, from Windows OS, it is possible to see when opening a hive, a key or a sub-key that each hive, key or subkey is represented by three fields, the *Name*, the *Type* and some *Data* regarding the Hive, the Key or the subkey.

As for the type field represents the type of value stored in that hive, key or subkey. Following on, is a list of the most common *Types* which can be found on the System Registry.

- REG_BINARY - The value stored on the variable will be binary. Usually this type is set to hardware component entries.
- REG_DWORD - These values are usually boolean, commonly a 0 is stored for a disabled option and a 1 is stored for an enabled option.
- REG_SZ - Usually the variables with this type store a string usually terminated with the NULL character.
- REG_MULTI_SZ - As for this type, usually stores more than one string (each one terminated with a NULL character), following no order. Usually this lists of strings terminate with an extra NULL terminator character.

Appendix B

Tinba/Zusy Basic Static Analysis

This annex intends to provide the basic static analysis to the `Tinba/Zusy` malware sample. On this annex all the information obtained from the enforcement of the earlier exposed tools and techniques in Chapter 5.

The same approach that was taken to the `Stuxnet` malware sample in Chapter 5 was taken towards the `Tinba/Zusy` malware sample.

As before, the first step took when presented with the sample was to run the sample on the VirusTotal website [tea] in order to discover if the potential threat has already been discovered and catalogued on any anti-malware solution.

As can be seen in figure B.1 almost all of the most important anti-malware solutions flag the file as malicious. But just like with `Stuxnet`, this was already expected.

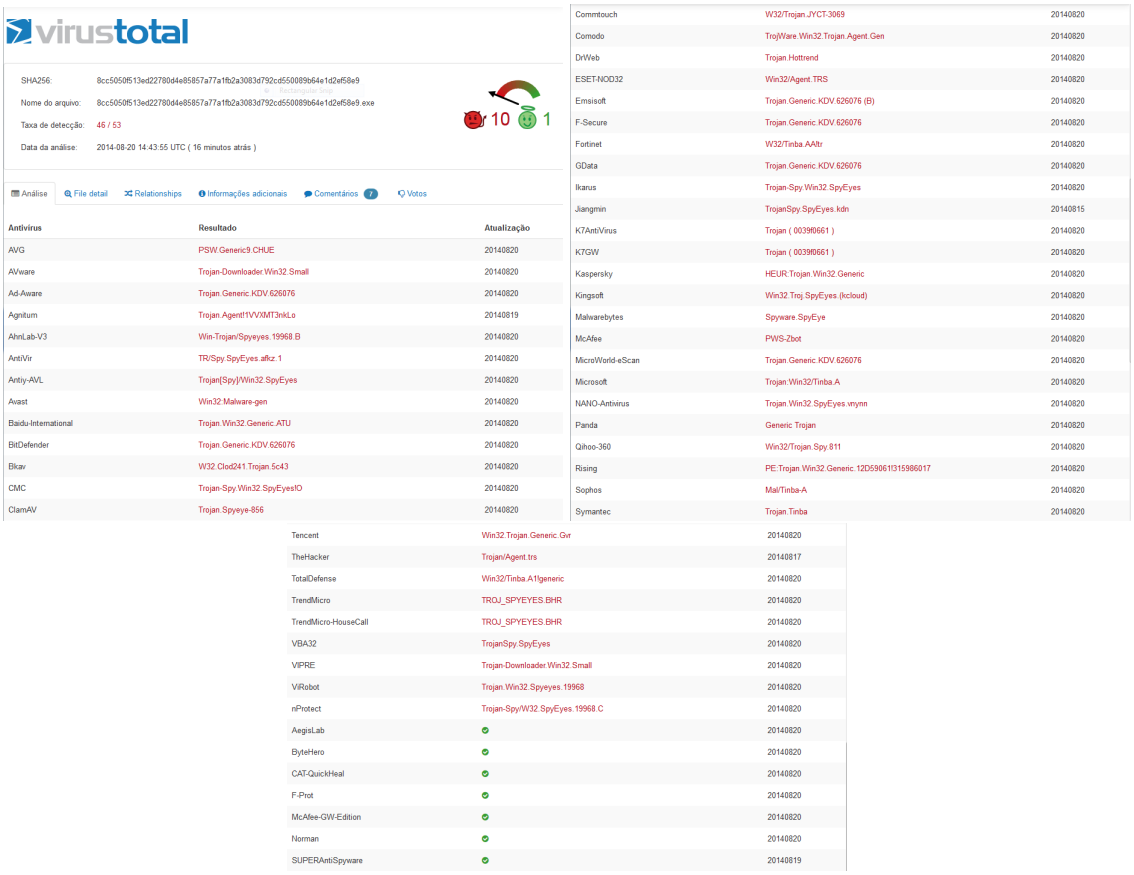


Figure B.1: Results of the Virustotal analysis on the `Tinba/Zusy` malware sample.

The second step is to find the hash value of the malware sample, using MD5. To obtain this value, WinMD5 [Win] was used, and the result can be seen in figure B.2.

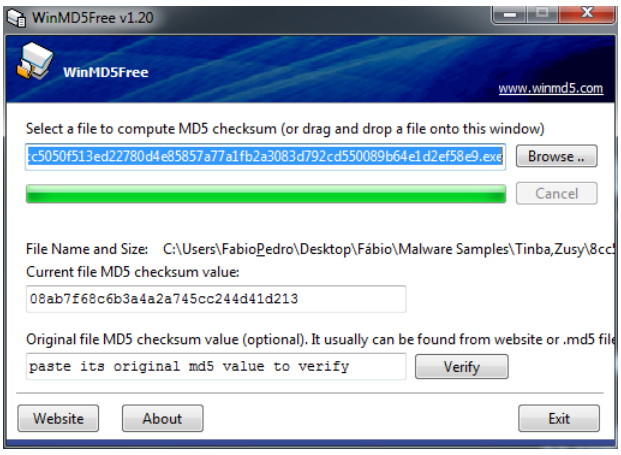


Figure B.2: WinMD5 result on the `Tinba/Zusy` malware sample.

This value as already explained can then be used to label the malware, use as an anti-malware database signature, send to another malware analyst in order to inform him that that signature might represent malware and even to use on an online search to try to find information on that signature.

Continuing on with the analysis, on figure B.3 some of the most important strings found on the `Tinba/Zusy` malware by recurring to the Windows SysInternals Strings software [Rus13] are presented. The strings depicted in the figure, had been precisely selected, and therefore, only

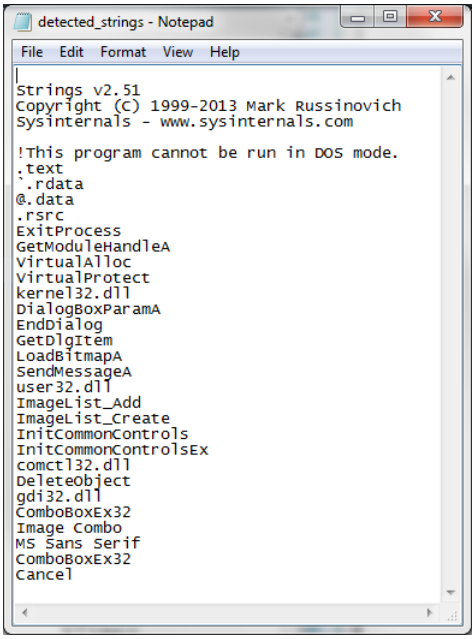


Figure B.3: The result of applying the Strings tool to the `Tinba/Zusy` malware sample.

show the strings which are able to provide some information to the analyst. The first 4 strings

shown, represent the PE sections of the malware piece while the others present a few methods that are probably called during the execution of the PE malware. Regarding the functions which can be identified under the strings, it is important to focus on 3 of them, which are described as follows:

- `GetModuleHandleA` - This function is called usually when wanting to obtain a module already loaded into memory. Malware usually uses this function in order to locate and modify loaded modules code, or to search for a good place to perform Process or DLL injection;

- `VirtualAlloc` - is a memory routine which is commonly used to allocate more memory on a process. Malware tends to use this function as part of the process injection behaviour; and

- `VirtualProtect` - This function is used to change the protection on a specific region in memory. Malware may use this routine in order to change a read-only section of memory to an executable.

From the three mentioned routines, the behaviour of injecting code into existing processes may be considered in further stages of analysis. However, it can also be seen by looking at the PE strings that it maybe presents a GUI to the user when executed, given the fact that there are a few strings which denote this, like: `ComboBoxEx32`, `DialogBoxParamA`, `EndDialog`, `LoadBitmapA`, and the API `gdi32.dll`. All of the aforementioned methods belong to the GUI common routines and APIs on Microsoft Windows OSs. So on further analysis techniques, when executing the malware, we might be presented with a GUI.

Continuing on with the basic static analysis, a search to find out if the threat is either packed or obfuscated must be done. To perform it, PEiD software [Sna08] was used. In figure B.4, can be seen that on the place that was supposed to be the packer used to pack the malware, its the `MASM32/TASM32` string, however, this is not the packing algorithm but the compiler of the PE, so nothing can be concluded about the threat being packed or not.

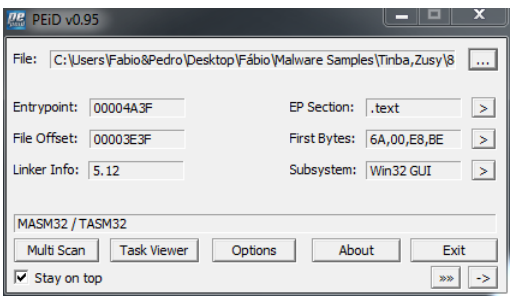


Figure B.4: Screenshot with the results provided by PEiD when analysing the `Tinba/Zusy` malware sample.

Now it is time to search for the imported and exported functions of the PE, which may prove

useful in understanding and preparing for the execution of the malware sample, once they may provide some behaviours which will then be noted when executing the sample. Figure B.5 shows the Dependency Walker software [Mil06] results when providing it with the `Tinba/Zusy` malware sample.

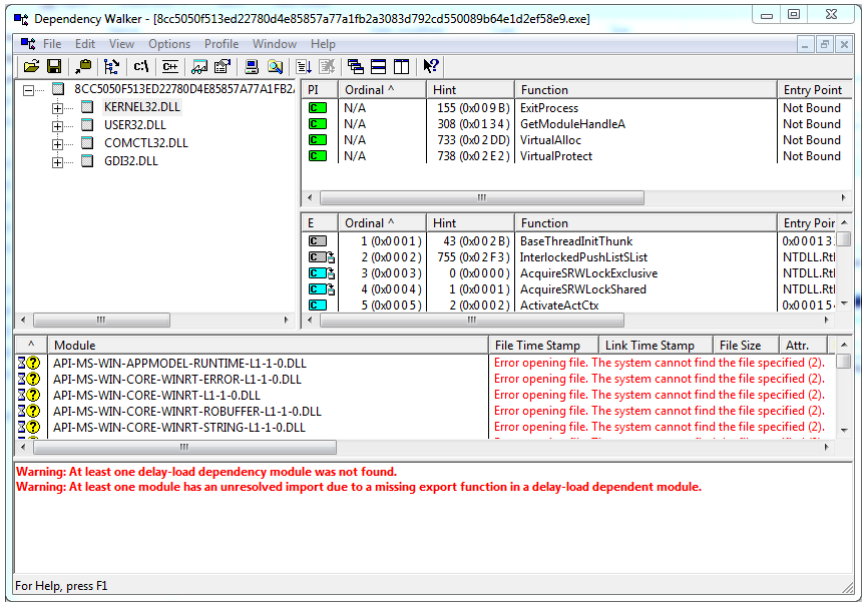


Figure B.5: Screenshot of the Dependency Walker application, showing the libraries imported by the `Tinba/Zusy` malware sample.

Even despite in figure B.5 the only shown routines are those from the `Kernel32.dll` (which are the most important towards this specific analysis), on the other three DLLs the functions called where the ones listed under the strings figure B.3 and already mentioned earlier on this annex.

The next steps where to find information under the *PEView* software [PEV14].

It was also possible to find that the compiling time of the PE file; which can be seen in figure B.6; it can also be noticed that the PE file is compiled under the GUI sub-system by looking into figure B.7; and that the file is probably not packed, once the *Virtual Size of Data* on the `.text` section of the PE file is even smaller than the Size of Raw Data by looking at figure B.8.

As for `Tinba/Zusy`, being the `.rsrc` section present, it was possible to obtain the bitmap and dialog resources by using Resource Hacker software [Joh11]. Those can be seen in figures B.9 and B.10 respectively.

As can be seen, no particularly good information can be drawn from this, except that the appearance of the possible GUI might be the one at figure B.10 and that the string *cancel* is probably the message that should appear on the Button of the DialogBox.

As for the `Tinba/Zusy` malware after performing the basic static analysis on one of its samples,

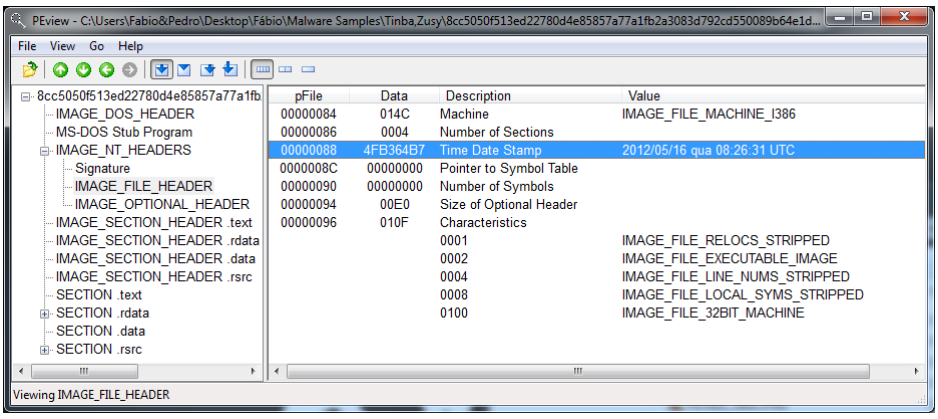


Figure B.6: Screenshot showing the `Time Date Stamp` field when analysing the `Tinba/Zusy` malware with PEView.

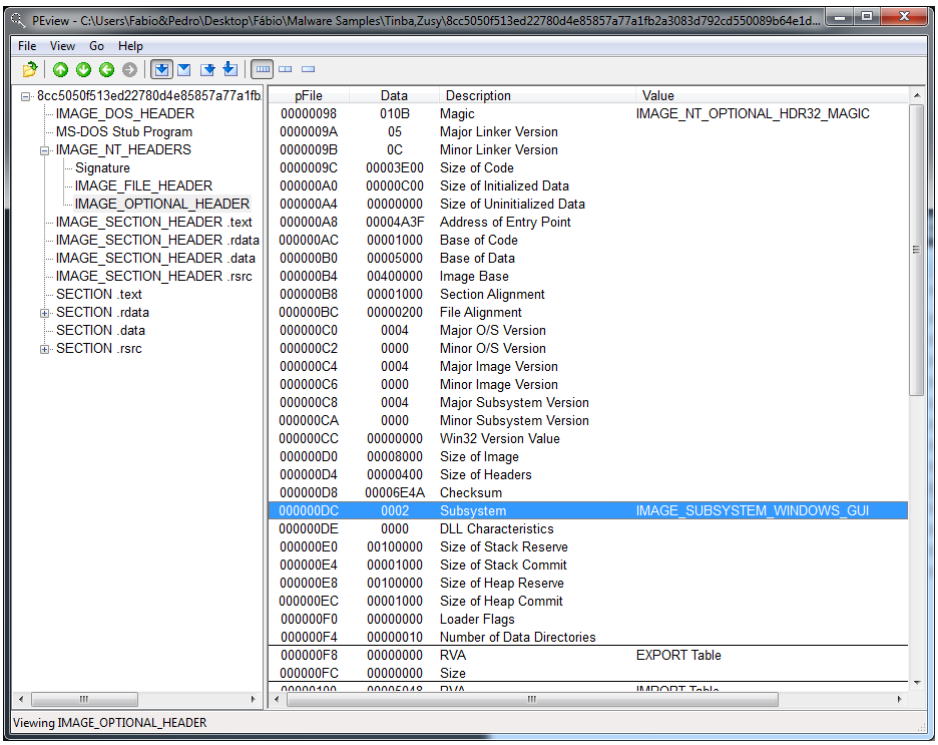


Figure B.7: Screenshot showing the contents of the `IMAGE_OPTIONAL_HEADER` of the `Tinba/Zusy` malware.

we were able to surely identify a few aspects which should be taken in account in further analysis steps, namely:

1. The PE is certainly malicious, once it has been flagged as malicious by 46 out of 53 anti-malware solutions on the Virustotal website;

2. The sample will probably perform any type of process injection or DLL Injection;

3. It might present a GUI to the victim in order to hide its real behaviour;

4. It probably is not packed which will turn it probably easier to analyse with more advanced

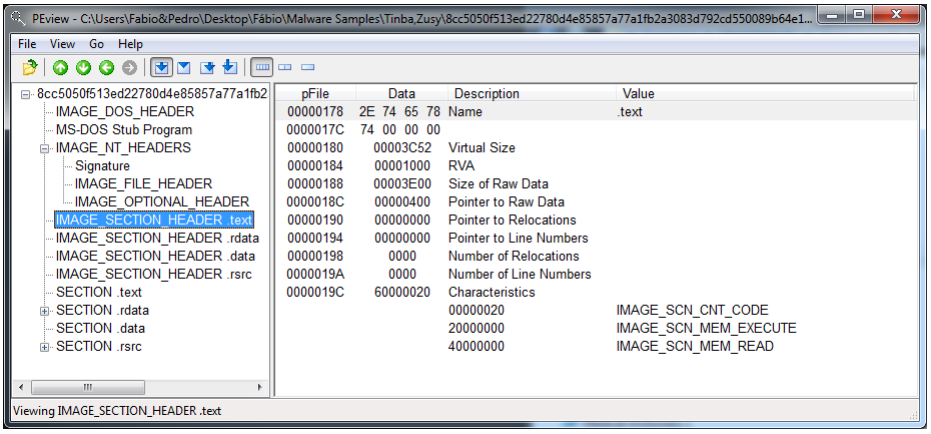


Figure B.8: Screenshot showing the contents of the `IMAGE_SECTION_HEADER` of the `Tinba/Zusy` malware.

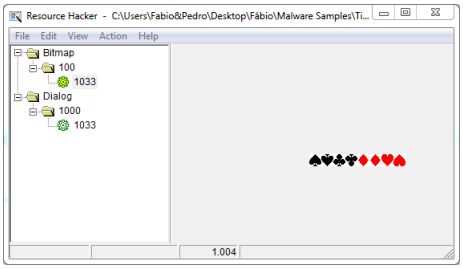


Figure B.9: Bitmap resources of `Tinba/Zusy` malware.

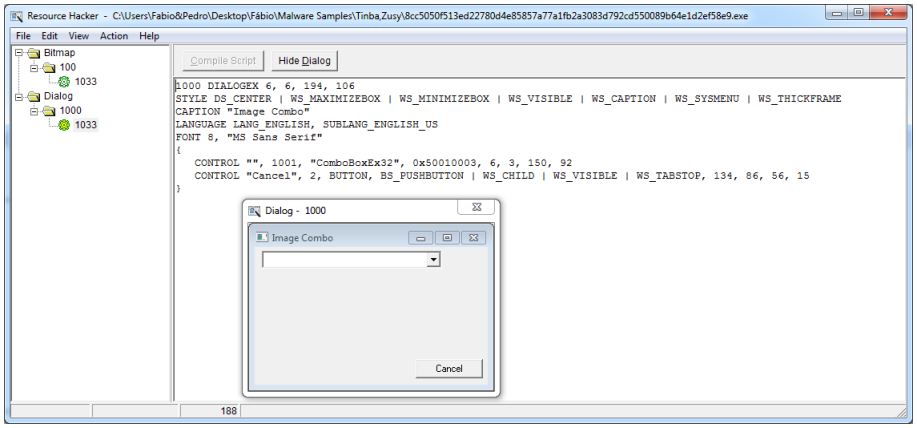


Figure B.10: Dialog resources of `Tinba/Zusy` malware.

techniques;