INTERNATIONAL
HELLENIC
UNIVERSITY

Malware Forensics Framework

Student Name: Provataki Athina

SID: 3301110017

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

OCTOBER 2012

THESSALONIKI – GREECE

INTERNATIONAL
HELLENIC
UNIVERSITY

Malware Forensics Framework

Student Name: Provataki Athina

SID: 3301110017

Supervisor:	Asst. Prof. Vasileios Katos
Supervising Committee Members:	Asst. Prof. Vasileios Katos Assoc. Prof. Nikolaos Bassiliades Dr Christos Berberidis

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Information and Communication Systems

OCTOBER 2012

THESSALONIKI – GREECE

Abstract

Cybercriminals today are able to orchestrate and realize massive or more targeted attacks using malware as the mean to invade and infect the victim's machines thus accomplishing their malicious intents. Detecting and analyzing such attacks might not always be feasible and could become a daunting and frustrating process. Targeted attacks are amongst the hardest to detect or analyze and pose a major security threat for organizations and large corporations as such malware attacks are extremely sophisticated and may go unnoticed for a large period of time magnifying the resulting damaging effects.

Modern malicious instances are characterized by composite behavior and functionality. As malware evolves and becomes more sophisticated malicious intruders have the ability to adjust their behavior depending on the infected system and its surrounding environment. Malevolent performance may be exhibited only upon the acknowledgment of specific system factors and the combination of several adjacent parameters and conditions. Certain behavioral aspects might be triggered upon the acknowledgment of specific environmental parameters while performance variances could differently affect each infected machine.

To overcome such shortcomings, we introduce a novel forensics methodology for assessing and reporting on the modus operandi of a malware in a specific organizational context. The proposed malware forensics framework facilitates multiple executions of the same malware in differently configured systems, in an automated manner, providing fast and inclusive results on how each malware behaves under a specific organizational context. The introduced analysis approach has the ability to correlate, analyze and interpret malware analysis results in an automated manner, significantly reducing time and effort needed to investigate and extract forensic intelligence information from a collection of analysis reports.

Student Name: Provataki Athina

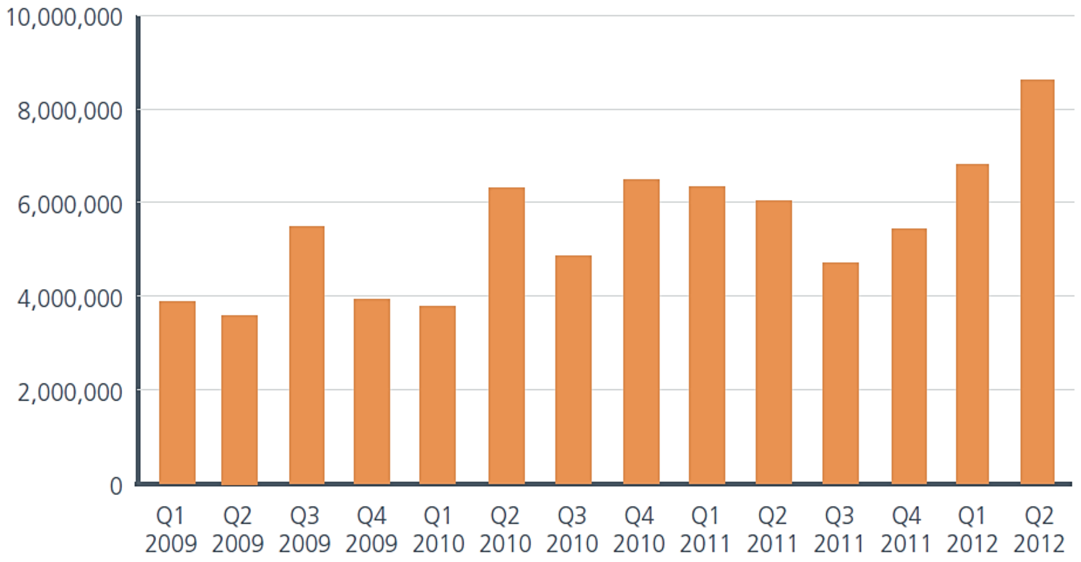
Date: 29-10-2012

Contents

1 Chapter 1 - Introduction

1.1 Overview

This section will contain an introduction regarding malware and malware forensics as well as the importance of malware analysis.



1.2 A Statement of the Problem

Every organization's goal is to build appropriate defenses in order to protect their company and prevent, as much as possible, any lurking intruder. Undeniably though, if criminals decide to attack they will sooner or later find a channel to achieve it. Cybercriminals today are able to orchestrate and realize massive or more targeted attacks using malware as the mean to invade and infect the victim's machines thus accomplishing their malicious intents. Detecting and analyzing such attacks might not always be feasible and could become a daunting and frustrating process.

As malware evolves and becomes more sophisticated malicious intruders have the ability to adjust their behavior depending on the infected system and its surrounding environment. Malevolent performance may be exhibited only upon the acknowledgment of specific system factors and the combination of several adjacent parameters and conditions. For example a specific malware might unveil its behavior only when installed on a Windows 7 platform or when a specific piece of software is installed on the victim’s PC (like PDF Reader) and remain completely dormant in any other situation. Similarly it may reveal a portion of its behavior while parts of its functionality will remain hidden until certain conditions are met that will trigger additional activity. Efforts have been made in order to unveil trigger-based behavior [13, 28] but it has also been proven that it is feasible to impede such analyzers [34]. Additionally, the existence of the so called Logic Bombs or Time Bombs further encumbers the analysis process and despite the effort made in this area [27] this subject still remains a problem in malware analysis.

Signature based identification that Antivirus vendors follow phases some significant drawbacks regarding the vastness of malicious samples submitted for analysis as well as specific restrictions on analysis time and platform variances [23]. A common methodology is to allow the executable to run for a short period of time and on test environments based on commonly used operating systems and applications, thus possibly not extracting its full functionality. To overcome such shortcomings modern approaches look for more automated ways of generating signatures [93].

Targeted attacks that are most often and regularly employed by cybercriminals form one of the major security threats that companies and organizations have to face and counteract [22, 23]. Such sophisticated and well instrumented attacks might remain undetected for a prolonged period of time within the victims’ organizational boundaries, amplifying the resulting damaging effects.

Recent approaches yield towards finding more automated procedures to facilitate the analysis process. Several automated analysis tools have been developed, like Anubis [86], GFI Sandbox[1] and Cuckoo [54], with the ability to perform automated analysis in a controlled environment. Some of these platforms are web-based and perform the analysis online while others can be locally installed and integrated into forming more holistic and comprehensive analysis frameworks. However, commercial solutions, like GFI Sandbox, are often a luxurious choice since a single product license costs at least $15K. Whatever the case, with the appearance of such tools, malware constructors have also come up with ways to identify if such a tool is being used and remain inactive or hide their malicious intent [32]. Moreover utilizing on-line analysis tools also comes with

[1] http://www.gfi.com/malware-analysis-tool#overview

restrictions regarding privacy and legal issues on the information that might be disclosed to such a third party [19].

Automated dynamic analysis tools alone suffer from restrictions and cannot fully address the complexity and severity of a complete forensic analysis methodology [13]. Dynamic analysis involves executing malware samples in a controlled and isolated environment to monitor and record the observed functionality [23]. However, a single execution of a given malware can only reveal a portion of the sample's behavior, relevant to the specific environmental conditions of the analysis run. Therefore different executions of the same malware could produce diverse analysis results.

Most of the available automated analysis tools work in a similar manner, performing the analysis and then producing a detailed human readable report containing the results for further processing. Even so new procedures need to be instrumented in order to provide the means for more automatic ways of processing a large number of such reports. Latest methodologies approach this issue by first submitting a number of malicious files for analysis, collecting and storing the reports in a database and later on performing differential analysis on those reports with the purpose of identifying either differences or similarities in the malwares' behavior [91, 26], focusing mostly on providing new classification and clustering mechanisms rather than revealing their full functionality.

To overcome such drawbacks and shortcomings, this dissertation introduces a novel malware analysis approach that automatically correlates and processes the analysis results amongst multiple executions of the same malware sample. The proposed malware forensics framework provides the means for assessing malware conduct and reporting on how a specific malicious sample behaved under a certain organizational context.

1.3 Academic Research Question - Aims and Objectives

Our research focuses on investigating the feasibility of developing an approach for assessing and reporting on the modus operandi of a malware in a specific organizational context.

The aim of this master thesis is to study and develop a methodology for analyzing malware conduct in a certain organizational context. More specifically, our goal is to develop and describe a malware forensics framework that will allow forensics analysts to understand and recognize how a specific malware behaves in a certain environment apart from its apparent and recorded behavior.

Our objectives, include describing the challenges in malware forensics, understanding the practices used in malware analysis, constructing a test bed for empirically evaluating malware and developing an analysis methodology for assessing malware conduct by identifying behavioral differences and similarities amongst multiple executions of the same sample in variant environments.

1.4 Research Methodology

An appropriate Literature review will initially take place, in order to gain a better understanding of recent malware analysis approaches and techniques as well as to identify the current challenges that hinder malware forensics methodologies. By studying how malware has evolved over the years we will be able to realize the sophistication and complexity of modern malicious instances that enable the deployment of advanced, highly profitable and destructive cyber-attacks.

Exploring a malware's complete structure and functionality is still a matter of ongoing research, as malware authors are constantly aware of modern malware analysis advancements and continually develop new techniques to defeat detection mechanisms and obstruct the analysis process.

Moser et al. [13] report that one possible solution to exploring and analyzing trigger based malware behavior would be to execute the malicious sample multiple times in various environments, in order to uncover and expose any diverse and variable malicious behavior. Therefore by examining multiple execution paths results in a more holistic view on the behavioral aspects of a specific malware.

In order to achieve the desired goals of this dissertation, a controlled environment, where malware samples will be tested among different systems, will be developed and configured. The Test Bed, which will be used for collecting primary data for analysis, will include Linux as Host and various guest systems operating in Virtual Machines. Offensivecomputing.net will be utilized as our malware pool, in order to acquire multiple instances for testing and experimenting. Using Cuckoo as our main malware analysis tool, each virus will be executed and tested in various systems. Malware behavior will be recorded and analyzed using both static and dynamic analysis techniques.

For each malware the analysis results among the different environments will then be collected and analyzed. This analysis will be performed using two different methodologies. The first one includes the empirical and manual observation of the malware conduct by identifying dropped files, created processes and API calls, as well as network activities. The second approach refers to a more automated analysis procedure. An appropriate methodology will be introduced that will automatically correlate and process the multiple analysis results, to extract significant behavioral artifacts and characteristics. This will be accomplished by identifying possible differences and similarities between the variant analysis results. Based on these comparison results we will be able assess malware conduct and generate comprehensive malware profiles corresponding to the overall exhibited functionality of each given sample.

Upon the completion of this research, a malware forensics framework will be developed and described.

1.5 Significance of Research

The proposed malware forensics framework facilitates multiple executions of the same malware in differently configured systems, in an automated manner, providing fast and inclusive results on how each malware behaves under a specific organizational context.

Our methodology utilizes open source tools and automates the analysis process of generated reports by tools like Cuckoo. This means that a malware analyst can submit a sample for analysis multiple times and run “Profiler”, the core component of our framework, to quickly get an insight on the sample’s behavior, starting at an abstracted level and diving deeper into more specific behavioral characteristics.

The introduced analysis approach has the ability to correlate, analyze and interpret malware analysis results in an automated manner, significantly reducing time and effort needed to investigate and extract forensic intelligence information from a collection of analysis reports. Furthermore the produced results are stored in both human and machine readable formats so as to enable further processing and investigative activities.

Even though our goal is not to detect deceptive activity, differences in a sample’s behavior might indicate malicious intent [25]. To this end, our framework may be also utilized as a malware identification tool that can raise analysts’ attention towards performing a more thorough investigation on a malicious specimen, upon the detection of possible behavioral differences.

The overall architecture and flexibility of the introduced forensics methodology in combination with the significant intelligence information that it can produce, has a major impact in the whole computer enabled society, since malware attacks may target any possible individual user as well as organizations and even entire nations.

1.6 Dissertation Structure

The contents of each chapter will be outlined and described in order to provide the reader with a holistic view of this dissertation.

2 Chapter 2 - Literature Review

This chapter includes an important literature review related to our topic of research as well as a discussion around malware evolution. A survey regarding significant academic and research work relevant to malware and malware forensics as well as malware analysis tools, techniques and methodologies is imperative in order to gain insight on recent advances and challenges associated with malware forensic analysis. Some significant definitions and dimensions regarding malware, malware analysis, malware forensics, methodologies and tools are presented and described.

2.1 Overview of Related Academic and Research Work

Modern malware analysis approaches seek to address the problems of malware detection and analysis introducing novel techniques and more comprehensive analysis frameworks. On the other hand, malware creators monitor analysis related research advancements and constantly manifest new mechanisms to hinder and evade analysis processes. The vastness of malicious samples that are introduced each day as well as the increasing complexity and intricacy of malware code further thwart antivirus researchers and analysis procedures. This has led analysts and malware authors into a continuous “*arms race*” on the exhibition of power and skills [23].

Malware analysis methodologies and related efforts can be generally classified into four main areas of research:

- Developing stealth and transparent analysis frameworks.
- Detecting analysis evasion mechanisms.
- Improving the efficiency of dynamic analysis techniques.
- Creating clustering and classification methodologies.

Modern malware creations encompass sophisticated mechanisms and anti-analysis characteristics that can significantly thwart analysis procedures and delude analysis results [32]. Analysis-aware malware has the ability to modify its behavior during execution or to remain completely dormant, hiding its malicious functionality upon the acknowledgment of an analysis environment. To address the problem of evasive malware, recent approaches focus on developing stealth and transparent analysis frameworks that prohibit malicious creations from detecting the analysis environment.

Frameworks like Cobra [81] and Ether [82] have been specifically designed so as to combat analysis detection mechanisms. Cobra performs dynamic malware analysis using stealth localized executions, by splitting the malicious code into segments and separately inspecting their instructions before execution. Every suspicious detection-enabled slice is replaced with a safe implant so as to protect the analysis process. Ether, on the other hand, uses hardware virtualization extensions to eliminate any guest analysis modules, susceptible to identification, and to remain undetectable by malware. However, both approaches suffer from increased performance overhead especially when fine-grained analysis is required [25]. Moreover, as stated by the authors themselves, Ether is not destined to perform real-time analysis and Cobra's performance is affected by the required interactive analysis.

Modern analysis methodologies lean towards identifying differences in the malware behavior focusing mostly on creating new and more effective detection and analysis methodologies or discovering previously unnoticed evasion mechanisms and anti-analysis techniques [24, 25, 39].

Johnson et al. [24] introduced a differential slicing methodology, which given a target difference between two executions of the same program, in either the same or different environments, tries to identify the specific environmental parameters and input differences that led to the noticed behavioral change. Their approach seeks to automate and facilitate the process of analyzing observed differences between two runs of the same program and understanding their root cause, thus allowing analysts to identify possible vulnerabilities or programs that exhibit different behavior among different systems. In contrast to our method, which processes the dynamic analysis results as they are produced by Cuckoo analyzer, the differential slicing algorithm works directly on binary programs and uses TEMU (a component of the BitBlaze framework [30]) as an execution monitor, in order to record execution traces for each program run, which are further processed aiming at producing a final causal difference graph. Furthermore, our approach examines multiple executions of the same malware with the purpose of identifying as many as possible behavioral differences under a specific environmental context, whereas Johnson et al work on a given known difference and focus on discovering its cause.

Balzarotti et al [25] proposed an approach aiming at discovering malware samples that employ various anti-analysis techniques to detect whether the underlying execution environment is an emulated test system. To achieve this, the malware sample is executed in an emulated environment as well as in a virtualized reference host and the sample's behavior between the two runs is then compared. The comparison is performed by recording the malware's interactions with the host during the normal run and then replaying the execution in the emulated environment using the same input parameters. Based on the perception that upon the same input arguments a program's behavior should be identical, any observed behavioral difference is conceived as an indication of a detection mechanism that led the malware to execute a different path. However, this approach suffers from various limitations as not all possible system calls can be recorded and replayed. Moreover, malware interactions that use multiple processes and random input numbers cannot be recorded and replayed. As opposed to our implementation which uses Cuckoo analyzer [54] and virtualized guest systems, the proposed analysis environment uses emulation technology and is based on Anubis [86]. Furthermore, the fact that the reference host is virtualized while the analysis platform is emulated, allows malware to manipulate and change its behavior simply by being run in such differently configured systems. The basic distinction between our approaches, apart from the implementation differences, is that, in contrast to our goal, the main purpose of their methodology is to detect analysis aware malware.

Chen et al. [39] introduced a novel methodology to deter malicious programs from infecting production systems. Based on an extensive research, a comprehensive taxonomy of possible evasion mechanisms actively employed by malware authors was created. Their approach was based on large scale malware executions among virtualized environments, uninstrumented machines and debugger implementations, to identify possible behavioral deviations, obtained by execution traces' comparisons. By creating fingerprints of the observed characteristics, they were able to imitate them on real machines so as to appear as instrumented, thus preventing malware infections. Even though the recommended methodology attempts to identify possible behavioral differences, in contrast to our framework, Chen's approach focuses on preventing possible system infections rather than identifying and understanding the behavioral plurality of malware samples.

Another major category of malware analysis approaches, relates to identifying and investigating all possible behavioral characteristics of a given malware instance, thus leading to more efficient and comprehensive analysis frameworks. Static analysis benefits from the possibility of unveiling and uncovering the complete malware's structure. However its prospects are often limited by extremely sophisticated obfuscation and packing mechanisms [33, 66, 74]. Dynamic analysis remains protected against such techniques since the malware's behavior is recorded during execution. Nonetheless only a single execution path can be analyzed, thus possibly not capturing any trigger-based malware behavior [13].

Bayer et al. in 2006 presented TTAnalyse [12], a tool for dynamic malware analysis. TTAnalyse uses Qemu, a PC emulator, to execute unknown binaries, restricted to Windows executable PE files, in a Windows XP SP2 emulated environment and generates a report containing analytical information regarding the sample's behavior and functionality. Through the process of monitoring calls to native Kernel and Windows API functions, as well as performing function call injection, it can identify Windows Registry and file system alternations, trace interactions with various system processes and log all respective network activity.

Even though TTAnalyse provides an automated malicious code analysis environment with fairly precise and accurate results, it still suffers from the monotony of the underlying emulation environment and the restrictive nature of the malware types that it can decompose. Furthermore, dynamic analysis alone cannot provide a holistic view on the behavioral plurality and diversity that modern malware may exhibit. The cause is that only one single execution path can be explored each time the analysis is performed [13].

This means that malicious actions that are activated under certain conditions, for example on a specific date like the Michelangelo virus, upon the existence of a specific file or with internet connection availability, cannot be observed and recorded. To address this later problem Moser et al. in 2007 extend their previous work and introduce a system with multiple execution paths exploration capabilities [13]. To achieve this, the proposed system traces critical input values that the harmful code reads and identifies key points during malware execution were control flow decisions depend upon those values. Whenever such a key point is spotted, a snapshot of the program's execution present state is taken before it is allowed to continue running. As the analysis proceeds, by returning to the captured snapshot for every identified conditional branch, requisite input values are manipulated allowing different execution paths to be activated. Even though this approach provides a more comprehensive assessment on malware's behavior, it also faces some drawbacks regarding the analysis time which can be hindered by dead code insertion as well as the fact that complexity might grow exponentially in reference to the number of possible conditional branches that might exist. Compared to our approach, the malware's behavior is captured through multiple executions on differently configured systems, rather than manipulating specific input values. This means that our system remains unaffected by increased complexity and obfuscation techniques.

Similarly to Moser's approach Brumley et al. implemented MineSweeper [28], a system which employs mixed and symbolic binary execution in order to identify and investigate trigger based code paths in an automatic and iterative manner. A possible issue with MineSweeper, as the authors' state, could be that their system might not be able to explore all possible and diverse branches. BitScope [29] also uses symbolic execution with the purpose of completely uncovering all possible aspects of the malware's behavior. A more general and holistic approach towards automated dynamic binary analysis can be found in the BitBlaze framework [30] which integrates both MineSweeper and BitScope, and along with a variety of components implements both static and dynamic analysis and might also be used as an automated analysis tool. Even though the aforementioned methodologies can provide more inclusive analysis results, recent work has shown that trigger-based behavior analysis approaches can also be impeded by the malicious authors [34].

Since antivirus vendors and security analysts are daily overwhelmed by massive amounts of new malware samples, efficient analysis and signature generation methodologies often become frustrating and daunting processes with possible ambiguous and incomplete results [23]. Recent approaches implement clustering and classification mechanisms in order to minimize analysis effort and time by reducing the number of samples that require extensive and thorough research.

Rieck et al [91] introduced a framework that enables automatic malware analysis using machine learning techniques. Their methodology allows the detection of malware behavioral similarities. Malicious samples that exhibit similar behavior are grouped into clusters which are then used to classify malware with unknown functionality. The malware samples are executed in a sandbox environment. The generated sequential reports are then correlated with behavioral patterns and machine learning methodologies are applied to identify new or known classes. Similarly Bayer et al. [26] implemented a behavior-based malware clustering methodology by extending the Anubis [86] dynamic analysis system. Their approach seeks to identify subsets of malware with similar behavioral profiles which are then used as primary data for their clustering algorithm. As the authors state, their system might be affected by possible evasive mechanisms and miss capturing specific trigger-based behaviors.

Perhaps closest to our approach is the methodology proposed by Martignoni et al. [80]. Similarly to our technique, they aim at improving behavior-based analysis procedures by producing more comprehensive results though through a cloud-based implementation. The proposed system architecture allows multiple malware executions of the same sample in differently configured systems. To achieve this, the malware samples are distributed to various end-users' machines with variable configurations and perform an in-the-cloud analysis sharing the computational power and recourses of the underlying analysis lab. The analysis results are then merged to produce the resulting malware profile. Even though the analysis concept and goals share many similar characteristics with our framework, the implementation technique and resulting outcomes are quite different. As the authors' state they "*have not yet addressed the problem of correlating the results of multiple analyses*". This means that the multiple analysis outcomes are not further processed and associated so as to produce a comprehensive behavioral malware profile. Thus, differences or similarities in malware behavior cannot be determined. Despite the fact that each malware is executed multiple times, the analysis is preferably terminated upon the acknowledgement of malicious behavior. To this end, the proposed system can be utilized as a promising malware detector since it can identify malware that might intentionally delay the exhibition of malicious activity or might enclose trigger-based execution conditions. Moreover, the introduced methodology faces significant limitations concerning security and privacy issues on the information disclosed to external users as well as the lack of stealth analysis environments against possible malware evasive mechanisms.

2.2 Historical Review

Malicious software and more particularly viruses and worms have the ability to invade and attack computer systems by attaching themselves into the infected host through the process of self-reproduction and multiplication [6]. Iliopoulos et.al [44] in 2008 compared malware evolution to the Darwinian evolution model resembling also the mutation and replication capabilities of viruses and worms to biological viruses. This self-replication mechanism though, that now defines and characterizes modern viruses, was born in academia and was investigated and researched by scientists more than six decades ago [11], long before the idea of a virus even existed[2].

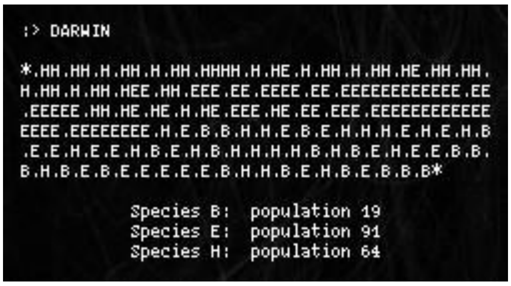
John Von Neumann was the first to illustrate the idea of self-replicating machines while giving a series of lectures at the University of Illinois, in the late 1940s, about the "Theory and Organization of Complicated Automata". In his lectures, Von Neumann compared the human brain and the human nervous system to computers with respect to volume, size and complexity elaborating also on the capabilities, hierarchy and evolution of complicated artificial automata as well as exploring and describing the possibility of designing a self-replicating computer program. His work, which constitutes the first ever academic approach to the theory of computer viruses, was published some years later, in 1966, as the "Theory of Self-Reproducing Automata" [1] and was implemented in practice almost three decades later [10], in 1995, by Umberto Pesavento who demonstrated a functioning simulation of Von Neumann's machine [4].

Following the work of Von Neumann, who investigated the logical conditions of the self-replicating problem and concluded in 1951 that it was possible to create a machine with self-reproduction properties, Lionel Penrose, with the help of Roger Penrose, approached the mechanical aspect and complexity of the self-reproduction theory. In his 1959 report called "Self-Reproducing Machines" Penrose builds on the idea of design-

[2] http://www.securelist.com/en/threats/detect?chapter=105

ing and constructing simple units with the ability to self-multiply, transmute and onslaught computer systems [2]. Inspired by Penrose, Frederic G. Stahl in 1961 created an Artificial Universe in which creatures had the ability to crawl, eat and replicate themselves [3]. Despite the limited capabilities and memory sizes of computers during that period, as well as the lack of presence of any OS, Stahl, using machine language on an IBM type 650 system, managed to successfully demonstrate reproductive and mutation mechanisms in computer programs.

In August of 1961, three engineers from Bell Telephone Laboratories (V. A. Vyssotsky et al.) experimented further with the self-replication capabilities of computer code and created a programming game which they called "Darwin: A Game of Survival and (Hopefully) Evolution" [5]. In this game the players had to construct programs, or so called species, on an IBM 7090 mainframe which were loaded into the arena, a designated memory area. Each one of the species could perform specific functions such as to multiply and make copies of themselves in unoccupied memory locations, or to track down and destroy other species by exploring their vulnerabilities, terminating the respective program and taking over the arena. Darwin's Umpire defined the rules of the game and the goal was to devise the most fertile replicator that would kill all other species. A screenshot of the game[3] can be seen in Figure 2.2.1. Darwin later on evolved into "Core War", one of the earliest popular computer games [37].



```
:> DARWIN

*.HH.HH.H.HH.H.HH.HHHH.H.HE.H.HH.H.HH.HE.HH.HH.
H.HH.H.HH.HEE.HH.EEE.EE.EEEE.EE.EEEEEEEEEEEE.EE
.EEEEE.HH.HE.HE.H.HE.EEE.HE.EE.EEE.EEEEEEEEEEEE
EEEE.EEEEEEEE.H.E.B.B.H.H.E.B.E.H.H.H.E.H.E.H.B
.E.E.H.E.E.H.B.E.H.B.H.H.H.H.B.H.B.E.H.E.E.B.B.
B.H.B.E.B.E.E.E.E.E.B.H.H.B.E.H.B.E.B.B.B*

            Species B:  population 19
            Species E:  population 91
            Species H:  population 64
```

Figure 2.2.1: Darwin – The game

For the years that followed, numerous theoretical approaches were born as researchers and academics continued to study and experiment with the phenomenon of self-replication and mutation [8, 9], driven primarily by their interest in the emerging fields of Artificial Intelligence and Robotics.

[3] http://www.retroprogramming.com/2011/07/darwin-celebrating-50-years-of.html

It is of no doubt that the work of the aforementioned experts and scholars empowered huge technological advancements, shaped and transformed future scientific trends and inspired many forthcoming researchers to build upon, progress and expand their heritage. No one however could have foreseen during that period that their efforts would become a stepping stone and fuel up what has later on proven to be a new form of an epidemic outbreak to the modern computer society.

2.3 Malware Evolution

A historical walkthrough around malware evolution is imperative in order to understand their advancement and progress as well as their elevation of capabilities, that has resulted into the existence of extremely sophisticated malware that have the ability to adjust their behavior depending on the infected environment, to perform evasive techniques to avoid detection and furthermore to infect not only computers but also other hardware and electronic devices. Throughout almost 40 years of history, malware evolved and transformed into an advanced cybercrime and cyber-terrorism weapon. Based on a thorough survey on related research and academic work [11, 15, 37, 52, 58], the following sections provide a description of how malware has evolved over the years.

2.3.1 The 70's – Experiments and Games

Historians and scientists are still debating on the actual birth date of the first virus. Nonetheless the first approaches to viruses and worms were pitched off mainly for experimental and research reasons [11].

Bob Thomas at BBN technologies in 1971, while experimenting, created a program with self-replicating capabilities called "Creeper". Creeper, which is now identified as the first computer worm, infected nodes of the ARPANET[4], the precursor of today's internet, and spread throughout the network by creating copies of itself. Even though Creeper managed to crawl and populate enormously it had no malicious intent and simply displayed the message "I'm the Creeper: catch me if you can". As a countermeasure the "Reaper" was devised in order to track down and destroy Creeper copies inside the network. Due to its capabilities, some not only consider Reaper to be the first

[4] Advanced Research Projects Agency Network (ARPANET) was the US military computer network

computer virus to be found in the wild but also credit it as the first Antivirus product[58].

Some years later, in 1974, the so called “Rabbit” virus appeared, stalling computer’s performance by rapidly creating multiple instances of itself on a single system. It is not yet clear whether it was part of another experiment or intentionally designed to crash systems.

During the same period, John Walker created a game called “Animal”, designed to run on Univac 1100 systems, which would prompt the players with appropriate questions in an effort to predict which animal they were thinking of. In 1975, in the attempt to automatically distribute copies of the game, he developed the “Prevade” routine, which could independently explore all accessible directories. Animal was bundled with Prevade and upon execution created a copy of the running game to any directory found containing an outdated version or when the game was not present at all. Prevade is considered to be the first self-replicating piece of software in the wild, implemented as part of another host program [15]. Some researchers even refer to Prevade as the origin of Trojans [58].

Inspired by Creeper in 1979, two Xerox PARC researchers, John Shoch and Jon Hupp, were the first to invent the term “worm” while exploring the idea of distributed computing [59]. They created an experimental program with the ability to search for idle processors inside their company’s network. Their program could replicate and attach itself to the inactive computers utilizing their CPU time. Even though appropriate precautions were developed in order to control and contain any unpredictable growth, their worm somehow escaped safeguards crashing a significant number of machines. Their program, also known as the “Xerox worm”, had no malicious or harmful intent, but rather was created to assist and promote research advancements in the field of distributed computing. However it also revealed serious issues related to controlling and restraining worm expansion.

2.3.2 The 80’s – From innocent pranks to “accidental” outbreaks

As the popularity of personal computers started to raise so did the interest of virus creators who began targeting microcomputers[5]. While the first era mostly dealt with exper-

[5] Personal computers during that time were called microcomputers as opposed to the existing mainframes.

imentation in favor to scientific advancements and beneficial purposes, the new trend seemed to enclose no malicious intentions but rather focus on innocent pranks, annoying infections and exhibition of capabilities.

Malware authors continued to experiment, exploring previously unknown replication and attack mechanisms, leading to the first actual destructive outbreaks, which were caused primarily by unintentional programming bugs and accidents.

The first Apple-II virus, which spread through infected floppy disks, was written in 1982 by Rich Skrenta. Elk Cloner's infection mechanism entailed copying itself to the boot sector of floppy disks and was activated every time the computer booted from an infected disk, subsequently spreading to any other disk being used. Its payload included a symbolic message[6] which was released every 50th boot.

In 1984, Fred Cohen introduced the term "virus" for the first time, in his research work "Computer Viruses - Theory and Experiments" where he presented a functioning demonstration of a computer program with self-replicating abilities [49].

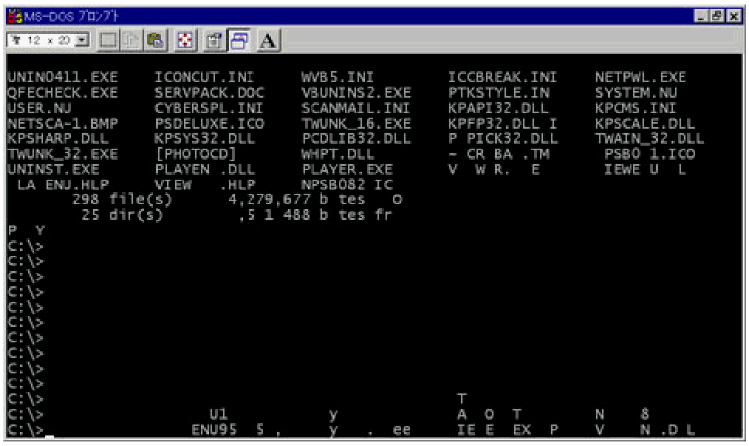
In 1986, the first PC virus that infected DOS-based systems appeared. Brain, presumably written by two Pakistani brothers, attached itself to the boot sector of floppy disks and had no harmful payload rather than labeling the disks as "© Brain" and displaying a simple advertising message. Brain managed to infect thousands of computers and presents the first instance of a stealth virus that effectively tried to hide its presence by displaying information irrelevant to the virus every time someone tried to access the boot sector's data.

For the years that followed viruses and worms continued to appear in the wild but their payloads remained quite innocuous and did not really cause any intentional damage to victims' computers. It seems though that during the late 80's the intentions of malware creators took a significant turn towards the development of more sophisticated malware samples that enclosed deceptive mechanisms and potentially disruptive payloads.

The first instances of actual destructive viruses were detected in 1987. The Vienna virus infected .COM executable files and is identified as the first one to curry a destructive payload. Once in every eight infections, the first bytes of the target file were replaced

[6] ELK CLONER: THE PROGRAM WITH A PERSONALITY IT WILL GET ON ALL YOUR DISKS IT WILL INFILTRATE YOUR CHIPS YES IT'S CLONER! IT WILL STICK TO YOU LIKE GLUE IT WILL MODIFY RAM TOO SEND IN THE CLONER!

with specific instructions that led to system reboot each time the program was executed, permanently damaging the file. Several Vienna variants emerged, mostly due to the fact that its assembly instructions were published in the book “*Computer Viruses: A High Tech Disease*”, written by Ralph Burger, while demonstrating how a computer can be infected by a virus [7]. The author also included a description of the Lehigh virus, which is regarded as the first .COM file infector virus with the ability to overwrite data residing on the disk. During that same year, the first virus with simple polymorphic features emerged. Cascade carried an encrypted payload to encumber disassembly and detection mechanisms. Figure 2.2.2 presents Cascade’s harmless payload of letters sliding down on the screen (adopted from [58]). Leigh and Cascade are considered to be an important milestone into the development of antivirus software.



```
UNIN0411.EXE    ICONCUT.INI     WVB5.INI        ICCBREAK.INI    NETPWL.EXE
QFECHECK.EXE    SERVPACK.DOC    VBUNINS2.EXE    PTKSTYLE.IN     SYSTEM.NU
USER.NU         CYBERSPL.INI    SCANMAIL.INI    KPAPI32.DLL     KPCMS.INI
NETSCA~1.BMP    PSDELUXE.ICO    TWUNK_16.EXE    KPFP32.DLL I    KPSCALE.DLL
KPSHARP.DLL     KPSYS32.DLL     PCDLIB32.DLL    P PICK32.DLL    TWAIN_32.DLL
TWUNK_32.EXE    [PHOTOCD]       WHPT.DLL        ~ CR BA .TM      PSBO 1.ICO
UNINST.EXE      PLAYEN .DLL     PLAYER.EXE      V  W R.  E       IEWE U  L
 LA ENU.HLP     VIEW   .HLP     NPSB082 IC
       298 file(s)      4,279,677 b tes   O
        25 dir(s)        ,5 1 488 b tes fr
P  Y
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>
C:\>                                             T
C:\>                  U1           y             A  O  T         N    8
C:\>_               ENU95  5 ,     y   .  ee     IE E  EX  P     V    N .D L
```

Figure 2.2.2: Cascade’s waterfall visual effect

The Christma Exec worm, that also made its appearance in 1987, marks the beginning of email spreading malware as well as the first instance of social engineering exploitation in order to lure IBM mainframe users into opening incoming infected emails. The worm, using Rexx scripting language, deceptively displayed a Christmas tree on the victims’ screen while in the background automatically emailed a copy of itself to the unaware users’ contact list obtained from their address books. The recipients were tricked into opening the infected message as it appeared to be sent by a familiar person.

Around late 1987 and early 1988, the first historically recorded targeted virus attack was detected. Initially, the Scores Mac virus seemed to have no payload at all, but further analysis of the disassembled viral code revealed that the virus looked for specific system resources that were later on identified to be part of the “EDS” company’s internal network [52]. Even though no actual damage was done, Scores carried complicated payload and trigger mechanisms. The first part of the payload was released upon infection and created two hidden folders and some notepad files, altering also their type and icons. The second one was triggered exactly four days after the attack. It searched specifically for “ERIC” and “VULT” running applications, and ended their execution after 25 minutes. The final viral part was activated seven days after the initial infection which caused a number of errors to the “VULT” related applications leading them eventually to crash.

Another destructive file infecting virus was discovered in the wild towards the end of 1987. Unlike the previous ones, the Jerusalem virus was the first MS-DOS based malware to attack .COM as well as .EXE programs. Its damaging payload though, did not manage to cause extensive losses because the virus was quickly detected due to an error which caused it to re-infect already infected files multiple times, significantly increasing their size.

Perhaps one of the most major security incidents and the first historically massively destructive malware outbreak took place in 1988 when the “Internet worm”, also named as “Morris worm” after its creator Robert T. Morris, managed to bring down thousands of Unix-based computers in just a few hours [53]. Unlike the Christma Exec worm, the Morris worm did not employ any deceptive mechanisms and did not require human interaction in order to spread. Moreover, it was one of the initial worm instances that employed a mixture of attack mechanisms to enhance its deployment. It could self-replicate, infecting one system after the other, by exploiting some already known unpatched networking and software vulnerabilities, such as in the Unix “sendmail” and “finger” daemon programs. In addition, by using a self-carrying dictionary of commonly used words in combination with any other dictionary possibly detected on the victim’s system, it attempted to break weak passwords and climb privilege levels.

Although the Morris worm caused approximately 100 million US dollars of financial damages, later enquiries inferred that the worm’s destructive behavior and outbreak were not intentionally provoked but rather a result of unpredicted parameters and programming faults. Such bugs also caused the worm to multiply uncontrollably, significantly reducing the performance of the infected machines, which ultimately led to its detection.

Towards the end of the decade, new mechanisms were developed, that transformed the previously exhibited virus functionality.

In 1989, a new arbitrary payload damage model was introduced by the Dark Avenger virus. Upon execution, the virus was loaded in memory and remained there subsequently infecting any other file being accessed by the user. Consequently, backup processes also became unreliable as data could get polluted and corrupted during copying attempts. Frodo on the other hand, which was discovered in Israel, was the first parasitic virus to enclose complete stealth behavioral characteristics. The virus tried to conceal the changes that its 4K code caused to the size of the infected files. Any information requests the users made regarding those files would display their initial size and not the increased one.

Possibly one of the first attempts that employed malware specifically for financial gain purposes was detected in late 1989 when a number of floppy disks, presumably enclosing AIDS relevant information, were distributed to the conference members at an international medical meeting. The packages came with a License Agreement that warned the users about the limited time period regarding the software's free use but it mostly got bypassed. The disks did indeed display some related material but the "AIDS" Trojan, as it was named, worked in the background and after a number of system reboots encrypted the hard disk's data and then presented a pop up message extorting the victims into paying the amount of $378.00 in return for the encryption key [52, 58]. The AIDS Trojan is also the first one of its kind that propagated using mailing lists and it is estimated that about 10,000 samples were dispatched worldwide to various medical institutions and other organizations.

2.3.3 The 90's – Polymorphism and Toolkits

By the beginning of 1990 users had become more aware of the risks involved and antivirus companies had already started to fight back by releasing new anti-virus tools able to detect more virus samples. The basic malware patterns and mechanisms had already been laid down and new malicious generations emerged mostly by building upon and extending these previously exhibited techniques.

During the next years, malware creators, following the current trends and developments, started to target also Windows-based systems and the first instances of macro viruses were developed. Simple encryption routines evolved into extremely complex polymorphism mechanisms raising new challenges in the anti-virus sector. The design of automated mutation engines along with the appearance of the first virus construction toolkits facilitated the massive creation of new malware instances. Furthermore, the expansion of the Internet as well as the adoption of e-mail as a preferred mean of communication provided new favored infection vectors for cyber attackers.

Earlier self-encryption techniques, aimed mainly at escaping byte pattern (signature) detection mechanisms that anti-virus products used for uniquely identifying viral pieces of code. Even so, the prepended decryption mechanism, responsible for returning the virus to its original form, stayed unmodified consequently allowing the malware's detection. In 1990 more complicated instances of polymorphic viruses began to appear in the wild. Attackers started to use complex encryption techniques such as transposing the virus's code and randomizing the decryption routine among different infections [11].

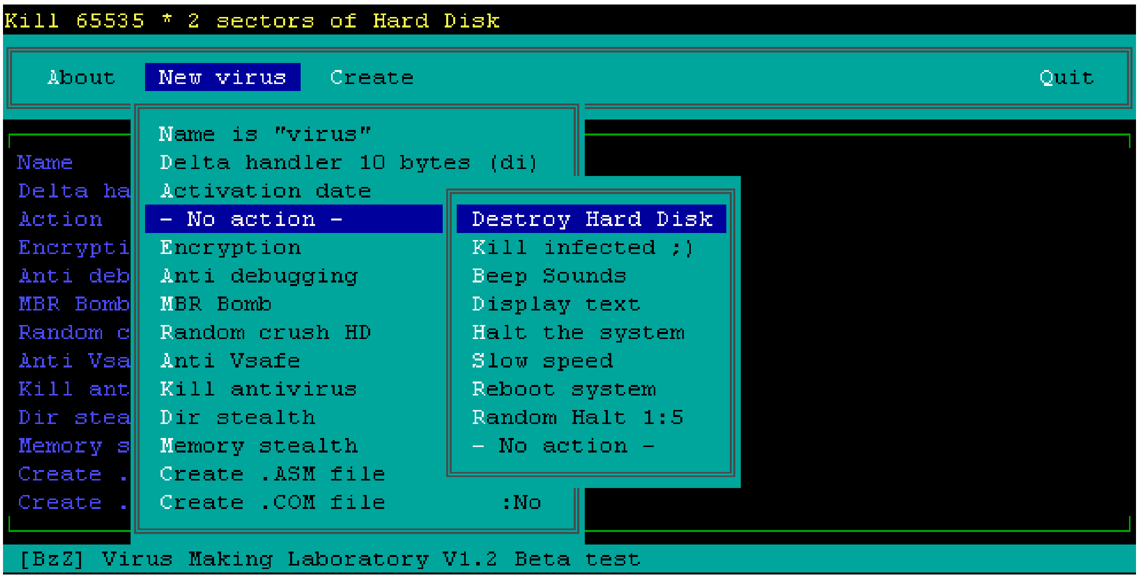
Amongst the first of such kind to appear was the Whale virus, discovered in June of 1990. This 9,000 bytes long virus employed innovative obfuscation techniques to avoid detection and to hinder disassembly and analysis procedures. It could recode itself between infections, constantly changing its appearance, making typical string scanning detection ineffective.

During the same period malware creators began establishing communities to better promote their goals. From such groups the first virus exchange bulletin board system (BBS) was launched, encouraging virus authors to upload new malicious programs in exchange for access to the system's virus source code database [58].

In 1991 the Michelangelo boot sector virus was detected. Its payload was triggered on every March 6th and could replace the first 256 hard disk's tracks with random data, destroying the system's boot information. Later that year Tequila, the first multipartite, fully polymorphic stealth virus entered the "battle field". Using advanced forms of polymorphism, such as variable encryption, it completely changed its viral code between infections, and could escape detection even from the best available antivirus software of that time [52]. By December 1991, around one thousand virus instances had already been identified.

The creation of polymorphic viruses required extensive programming skills. However, in 1992, the notorious Dark Avenger programmer developed a self-mutating engine (MtE) able to insert polymorphic characteristics to any virus [60]. With such a tool, malware coding and their polymorphic transformation became an easy task. Even though it facilitated polymorphic virus generation, any malware instance created with the MtE carried a unique signature, making it detectable. Soon after, new variations of mutation engines were released such as the Trident Polymorphic Engine (TPE), the Nuke Encryption Device (NED) and the Dark Angel's multiple encryption engine (DAME) [52].

In the summer of 1992 the virus creation landscape drastically changed. The first malware making toolkits emerged, enabling the massive generation of viruses through an easy menu driven interface [60]. The "Virus Creation Laboratory" (VCL) and the "Phalcon/Skism Mass-Produced Code Generator" (PS-MCP) that followed shortly after, developed by Nowhere Man and the famous Dark Avenger programmers respectively, provided amateur code writers with an already pre-constructed pool of viral codes to select from and simply apply their desired payload. Although such toolkits facilitated the creation of many different malware instances, their basic construction modules remained identical. Detecting one such virus subsequently meant that all others could also be identified [52]. Figure 2.2.3 illustrates a screenshot of the Virus Making Laboratory [58] that attackers could use to deploy new viruses without using any code programming at all.



```
Kill 65535 * 2 sectors of Hard Disk

  About   New virus   Create                                    Quit

 Name         Name is "virus"
 Delta ha     Delta handler 10 bytes (di)
 Action       Activation date
 Encrypti     - No action -          Destroy Hard Disk
 Anti deb     Encryption             Kill infected ;)
 MBR Bomb     Anti debugging         Beep Sounds
 Random c     MBR Bomb               Display text
 Anti Vsa     Random crush HD        Halt the system
 Kill ant     Anti Vsafe             Slow speed
 Dir stea     Kill antivirus         Reboot system
 Memory s     Dir stealth            Random Halt 1:5
 Create .     Memory stealth         - No action -
 Create .     Create .ASM file
              Create .COM file           :No

 [BzZ] Virus Making Laboratory V1.2 Beta test
```

Figure 2.2.3: The Virus Making Laboratory

Creating malware soon became a profitable underground business when constructors started selling their virus creations. For example the European Virus Clinic offered their malware samples for approximately $25 while another virus programmer was selling his creations for about $100 [58].

In 1995 the first macro virus, named “Concept”, appeared in the wild. Concept targeted Microsoft Word documents and quickly managed to spread worldwide. Macro viruses in general are easily created and platform independent. However, in contrast to more advanced modern creations, the initial instances of macro viruses could easily be blocked, simply by disabling the text processor’s macros [11, 52].

In 1997 the first virus targeting Linux-based systems was detected. Bliss infected linux executable files and exhibited worm spreading behavior locating potential victims through the */etc/hosts.equiv* trusted hosts list [58].

The appearance of the Melissa macro virus in 1999 designated the birth of a whole new generation of fast e-mail spreading malware. Melissa spread around through an infected word document as an e-mail attachment. Every time a user opened the e-mail, Melissa automatically retrieved the first 50 contacts from the user’s Microsoft Outlook address book and mailed itself to the recipients. This new mixed type of malware with worm-like propagation mechanisms, managed to infect thousands of computers globally in just a few hours crashing also a significant number of mail servers due to the increased volume of generated e-mails.

2.3.4 The 2000 Decade – Social Engineering and Cybercrime

During this new decade, malware creators’ exhibition of power and coding skills continued to surprise the computer security world. This new modern wave of malicious invaders is characterized by an advanced level of sophistication and complexity accompanied by more damaging payloads and destructive results. Malwares’ propagation speed rapidly increased and new forms of blended threats emerged combining multiple infection vectors concurrently [11]. Cyber criminals widened their attack range by targeting previously unthreatened platforms and network technologies, like Linux and Peer to Peer networks, as well as modern electronic devices such as mobile phones. Antivirus vendors not only faced the first attacks targeting antivirus software but were also challenged by new dynamic malware updating techniques.

The first attempts towards creating a network of compromised and controllable computers were realized and led to the evolution of Bots and Botnet technology [61] that facilitated the changing interests of cyber criminals. From noticeable data destructive and machine corruptive payloads attackers moved on to invisible and well hidden components aiming at covertly monitoring on line activities and stealing personal information and data that could then be utilized for internet fraud or other illegal financial gain purposes. Malware designers soon started selling or leasing their malware creations in the form of toolkits to any lurking cyber attacker [63]. The newly established underground malware market became a profitable business, opening the gates to novel criminal business models [64] and building an important foundation for the evolution and transformation of modern cybercrime [65].

Following Melissa's traces pretty soon new advanced fast e-mail spreading hybrid malware forms were developed. The BubbleBoy scripting virus and the KAK scripting worm that emerged in early 2000, revealed a new undiscovered security threat. The malware's viral code was not included as an attachment but rather executed automatically as soon as the victim opened the received e-mail message. Their creators took advantage of an Internet Explorer's vulnerability, which allowed them to insert an HTML document containing the malicious Visual Basic script inside the main message body of the e-mail [11, 52]. Similarly to Melissa, they then propagated throughout the network by sending themselves to e-mails harvested from the victims address books.

Worm designers continued to come up with innovative mechanisms to increase their victims range and enhance their destructive outcomes. The Love Letter worm for example tried to manipulate e-mail recipients into opening the attached "Love Letter" message, by prompting them to kindly read it. The attachment was actually a VB script and the file's name contained two extensions ".TXT.vbs" in the hope that it would pass unnoticed [52]. To further extend its attack vector the worm also created an IRC connection subsequently infecting anyone who participated in the IRC channel [11]. Love Letter's destructive and information stealing attitude caused up to 10 billion US dollars economic loses and is considered as one of the most damaging viruses in malware history [58].

The trend towards employing social engineering techniques to trick unaware users soon became a common practice among forthcoming massive e-mailing attacks while the engagement of IRC technology set the ground for the development of future Botnets [61]. On the other hand, the appearance of the Hybris worm opened the gates to another generation of sophisticated malware that can dynamically modify their viral structure and payloads by automatically downloading updated code versions.

The beginning of the new millennium also brought to surface probably the largest Denial of Service (DoS) attack up to that time. A young attacker known as the Mafiaboy, carried out a Distributed DoS (DDoS) attack targeting high profile websites like Amazon, Yahoo and CNN. To achieve his goals he took under his control several computers over the network and organized a large-scale Ping-of-Death attack[7]. Yahoo's services remained unavailable for approximately 8 hours costing the company millions of dollars in financial damages. However the attacker was detained for only a few months and got away with a $650 fine [58]. Even though modern networks are protected against Ping-of-Death attacks, other types of DoS attacks are feasible and continue to pose a great security threat.

Attackers soon started to target new communications devices such as the Timofonica virus which is acknowledged as the first one to infect mobile phones [58]. Even though its payload was harmless, it revealed a new movement towards compromising different gadgets other than traditional computer systems.

The global expansion of the Internet and the continuous development of advanced web services and web-based applications also brought along new security weaknesses that attackers soon began to exploit. Malware authors started to explore new infection vectors and propagation mechanisms in addition to the well established by now e-mail proliferation method, and payloads become even more destructive and complex.

In 2001 the Lion worm attacked Linux platforms by exploiting a buffer overflow vulnerability of the BIND DNS server. Once installed, it gathered passwords and other personal information and e-mailed them to its controller. Lion's complicated payload also installed backdoors, binary toolkits and a DDoS agent.

The SadMind worm targeted both Sun Solaris systems and Microsoft's Internet Information Services (IIS) web servers. It took advantage of a buffer overflow weakness to attack Sun systems. It then installed additional programs in order to infect Microsoft's web servers and coordinate website Defacement attacks.

The Code Red worm which also appeared in 2001, exploited the freshly discovered Index Server ISAPI buffer overflow vulnerability in Microsoft's IIS web servers. Immediately after infection, the worm created multiple duplicate threads responsible for attack-

[7] A Ping-of-Death attack entails sending ping packets larger than the maximum supported packet size in IPv4 networks (65,535 bytes).

ing additional IIS servers through a generated list of IP addresses. A programming bug caused the worm to create the same IP lists on every victim host significantly slowing its expansion. The Code Red v2 came to the rescue almost immediately to repair the previous bug. By correctly creating a random IP address list it managed to infect 359,000 systems in just a few hours [11]. The worm's payload carried out DoS and website defacement attacks against specific targets. The Code Red II variant that followed shortly after additionally created a Trojan and a backdoor and spread through an IP target list randomly generated from inside the host's subnet.

One of the fastest growing widespread blended attacks was realized also in 2001 by the Nimda worm which combined multiple infection vectors concurrently in order to proliferate, significantly increasing its complexity and propagation speed. The worm exploited a number of previously known vulnerabilities and infection vectors and spread by:

- Repeatedly e-mailing itself as an ".EXE" attachment to addresses retrieved from the system's cache and mailbox.
- Randomly infecting Microsoft IIS web servers using a buffer overflow vulnerability that allowed code execution on the server.
- Creating exact copies of the worm on network shares of a compromised server.
- Inserting Javascript into web pages stored on the computer.
- Utilizing backdoors created by the SadMind and Code Red II worms.

The Nimda worm took extra measures to avoid detection and its payload included modifying Registry and System files as well as creating shares and accounts with administrative privileges to enable remote access.

The first instances of malware targeting anti-virus products were also identified. The Bugbear and Klez worms were amongst the first to track down and kill any running processes generated by antivirus software as well as destroy any related files stored on the hard disk. The Bugbear additionally installed a Keylogger Trojan horse for capturing and recording any keyboard typing actions.

Probably two of the worst disastrous security attacks and massive worm outbreaks ever to have been realized took place in 2003. The SQL Slammer and Blaster worms achieved an incredibly fast infection rate spreading to hundreds of thousands of systems globally within just a few minutes. SQL Slammer carried no payload at all and rather focused on rapidly replicating across the Internet. It performed a buffer overflow attack on Microsoft's SQL servers and propagated between hosts through a single UDP packet. Blaster on the other hand targeted Windows XP and 2000 platforms by exploiting their DCOM RPC[8] system's vulnerability. Copies of the worm were dispatched to the victims' machines through remote calls on port 4444. The worm's payload installed a DoS agent and attempted to perform a TCP SYN flood attack against Microsoft's Windows update website [11].

Another important milestone in malware's advancement was set by the Sobig worm and its variants. The worm propagated through e-mails and is considered as the first coordinated effort towards creating a large network of infected computers that can be manipulated and directed remotely by a master controller. It is believed that the attackers tried to build a Botnet of zombie machines in order to perform massive Distributed Denial of Service (DDoS) attacks [58].

During the next years malware and more particularly worms continued to appear and spread throughout the globe evolving their infection mechanisms and widening their attack vectors. The Bagle worm for example, detected in 2004, was able to attack all Microsoft Windows versions while the Cabir worm was specifically designed to infect Symbian-based mobile phones replicating through Bluetooth wireless transmissions.

In early 2007 a new emerging threat was detected. The fast spreading Storm worm employed social engineering techniques to attract victims and propagated through e-mail attachments of various subjects, with the initial one pretending to deliver news regarding a weather catastrophe. The worm came with a dangerous payload which upon execution downloaded and installed other Trojans, Backdoors and rootkits in order to recruit computers into a huge Storm botnet. As it was reported, one of the infected systems was noticed to send out approximately 1,800 e-mails within only five minutes [58]. In contrast to other common botnets, the infected zombie machines were not centrally controlled but rather built on top of a Peer-to-Peer network technology. It is estimated that a few months after its release the Storm botnet controlled up to 10 million compromised machines. The resilience and complexity of the Storm makes it hard to detect and contain as it is repeatedly modifying its packing routine and uses fast flux technology to constantly change the IPs of the C&C servers[9].

[8] Distributed Component Object Model Remote Procedure Call

[9] Looking inside the Storm worm botnet, http://news.cnet.com/8301-1009_3-10009953-83.html

Botnets soon became a flourishing underground business for cybercriminals [64, 65]. Botnet creators started selling or leasing their bots allowing attackers to manipulate their zombie networks as they pleased [63].

The Zeus Trojan, which was first spotted in 2007, was specifically designed to create a botnet of compromised computers that could then be recruited for information stealing and financial fraud activities. It spreads through phishing and drive-by attacks and collects information using various techniques such as for example form grabbing and keystroke monitoring. It was estimated that by 2009 the Zeus botnet had already infected 3.6 million computers in the United States and was responsible for 44% of the known banking cyber-attacks [62]. Pretty soon Zeus circulated in the underground market as a crime-ware toolkit available for purchase with its price ranging from a few hundred dollars up to $15,000 depending on the malware’s version and added features/modules[10]. The toolkit comes with an easy installation procedure and customization mechanisms so that attackers can select the type of information they want to steal or gain access to such as banking account credentials, credit card details, e-mail accounts and any other type of personal data. Zeus uses complicated multi-level obfuscation mechanisms to avoid detection and hinder any analysis procedure. Researchers have only recently attempted to reverse engineer the Zeus toolkit in an effort to get an insight on its advanced technology and structure [62]. In 2012 new variations of ZitMo (Zeus’s version for mobile phones) were discovered specifically designed to attack Android and BlackBerry devices[11].

By 2009 the cybercrime landscape had completely evolved and malware authors were able to perform highly sophisticated attacks utilizing botnet technology and complicated malicious constructions. This led to the formation of a flourishing underground economy and to the emergence of highly profitable crimeware business models [71].

2.4 Latest Malware Attacks

Following the evolution of cybercrime, novel emerging threats formed a new trend towards advanced cyber-terrorism attacks. These latest types of attacks have revealed ex-

[10] http://www.secureworks.com/research/threats/zeus/?threat=zeus

[11] http://www.scmagazine.com/blackberry-android-users-targeted-by-new-zeus-trojan/article/253940/

tremely dangerous and frightening attempts with the potential to threaten not only explicit organizations and industries, but even entire nations.

In June 2010, Stuxnet, a new threatening cyber-attack was detected [70, 92]. The Stuxnet worm was the first one to specifically target SCADA[12] industrial systems and employ PLC rootkit technology that allowed PLC[13] code modifications. Even though it was initially found to target Iranian facilities, the worm managed to quickly spread to various other countries. The worm was designed to propagate through various zero-day vulnerabilities, such as previously unknown exploits on the print spooler service and Microsoft Windows Server services, as well as through removable media. Stuxnet features included stolen component certificates, complicated injection and hooking mechanisms, dynamic updates and specific antivirus evasion utilities. Its ability to reprogram PLCs and control such critical industrial infrastructure, position Stuxnet as one of the most dangerous cyber weapons ever to have been created.

In 2011 a new sophisticated worm attack was identified. Dugu[14] was specifically designed to steal passwords, collect computer screenshots and any other information located on the infected machine. Dugu spreads through a previously unknown vulnerability in Microsoft word documents and unlike Stuxnet, its main purpose is to conduct industrial espionage.

Flame, another emerging cyber threat, was discovered most recently, in 2012, by Kaspersky Labs[15]. Security experts announced that Flame was already in the wild for two years but managed to remain undetected due to its extremely complicated and sophisticated structure as well as the targeted nature of its attacks. Flame’s primary objective is to perform cyber espionage collecting any type of available information with abilities among others to intercept network connections and capture audio recordings. Similarly to Stuxnet, Flame can replicate through removable disks and the same spooler vulnerability. Latest researches have concluded that Stuxnet, Dugu and Flame are inter-

[12] Supervisory Control and Data Acquisition

[13] Programmable Logic Controller

[14] http://www.kaspersky.com/about/press/duqu

[15]http://www.kaspersky.com/about/news/virus/2012/Kaspersky_Lab_and_ITU_Research_Reveals_New_Advanced_Cyber_Threat

related[16]. Analysis processes revealed that Dugu and Stuxnet were created using the same platform while specific Flame modules were identified inside Stuxnet's code.

More recently, a series of "High Roller" banking attacks were revealed[17]. The attackers combined and customized three popular malware toolkits, targeting high balance bank accounts. SpyEye, Zeus and Ice 9 were employed to perform one of the most sophisticated, extremely automated and targeted banking fraud attacks. As it was difficult to breach through banks' security systems, cybercriminals infected the clients' computers. During the execution of on-line transactions, users were prompted with waiting messages allowing on-line robbers to steal the funds from the users' accounts by conducting automatic transfers. According to McAfee's report, the total amount of stolen funds is estimated around 2 billion US dollars.

The ease by which such attacks can be realized as well as the wide availability of preconstructed malware toolkits (for example the Zeus toolkit can be purchased in the underground market for about €1.000) and the extremely huge profits that can generated, has significantly contributed to the massive expansion and evolution of cybercrime activities and attacks [71].

While initial malware instances were quite innocuous and highly visible, their evolution gradually moved towards extremely dangerous, highly profitable and well concealed malware attacks. Figure 2.2.1 graphically presents the evolution of malware throughout the decades in terms of maliciousness, profitability and visibility characteristics [71].

[16] http://www.kaspersky.com/about/news/virus/2012/Resource_207_Kaspersky_Lab_Research_Proves_that_Stuxnet_and_Flame_Developers_are_Connected

[17] http://www.mcafee.com/us/resources/reports/rp-operation-high-roller.pdf

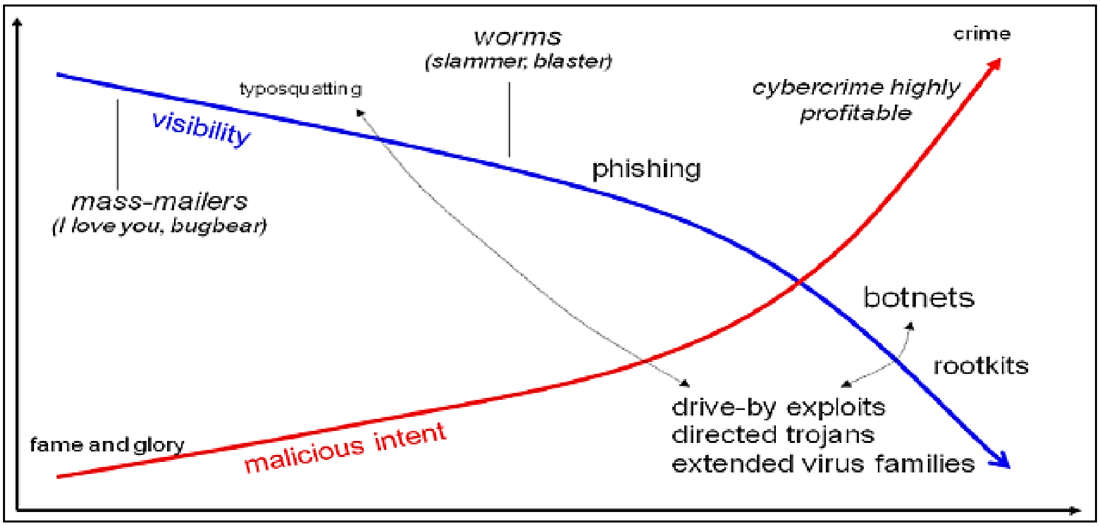


Figure 2.2.1: Malware evolution

2.5 Malware Types

Malware is a broad term and generally refers to any piece of software that intentionally performs malign activities. Moser et al. define malicious code (malware) as "*software that fulfills the deliberately harmful intent of an attacker*" [13]. Malicious software, depending on its purpose and functionality, is further classified into several behavioral categories. Terms like "Viruses", "Worms" or "Trojans" are used to describe malware samples with resembling behavior. This section provides a brief description of some of the most common malware types that can be found in the wild, in order to gain a better understanding of their characteristics and how each malware family functions and operates.

Even though we usually classify and refer to malicious code as a "virus" or a "worm" etc., it should be noted that a specific malware sample may not exclusively belong to only one class [23]. This is because many times the observed functionality of a certain malware might resemble the behavioral characteristics of multiple malware types concurrently. More details and information regarding malware types and malicious software in general can be found in [15, 22, 37, 52].

Viruses. The term "virus" was introduced for the first time in 1984 by Dr. Frederick Cohen who described a virus as "*a program that can 'infect' other programs by modifying them to include a, possibly evolved, copy of itself*" [49]. Even though the term virus is closely associated with harmful intentions and damaging results, according to Cohen's definition the main characteristic of a virus is its ability to self-replicate by infecting other programs. Therefore, if strictly interpreted, someone may conclude that any type of software with replicating abilities, even with no malicious payload, can be referred to as a virus [52]. Skoudis and Zeltser define a virus as "*a self-replicating piece of code that attaches itself to other programs and usually requires human interaction to propagate*" [15]. Consequently, a virus needs to infect and modify other files in order to replicate and spread.

In general, a computer virus cannot be executed autonomously. It inserts its set of instructions into the command chain of another program (host) so that when the host file is activated, usually by a user's intervention, the viral piece of code is also executed. Typical virus hosts may include [51]:

- Executable files usually disseminated through emails as attachments.
- Disk partitions' Boot sectors.
- Script files such as batch or shell scripts.
- Any macro containing document, such as Microsoft Office documents etc.

Depending on the target host, viruses can further be distinguished into "File Infecting Viruses", "Boot Sector Viruses", "Scripting Viruses" or "Macro Viruses" [52]. The basic structure of a computer virus usually contains at most three main subroutines or mechanisms [11, 52]:

- *Infection*: This part of the viral code defines the propagation methods of the virus.
- *Payload*: Specifies the actions to be performed on the polluted host.
- *Trigger:* Defines when exactly to release the payload.

The infection mechanism, unlike the other two, is always activated and defines how the virus will proliferate among possible candidate hosts as for example files of a specific type and content or whether to prepend or append itself inside the host file. Upon execution, the host also behaves as a virus and, based on the infection routine, copies the viral code into other programs thus enabling the virus's replication mechanism.

Moreover, the virus does not always simply create exact duplicates of itself. To further encumber detection, it may modify itself, for example by rearranging the sequence of some instructions, thus mutating as it spreads from host to host while preserving the same functionality (Metamorphic viruses). More advanced viruses might also encrypt themselves using different keys along infections while preserving the decryption algorithm (Polymorphic viruses). Viruses additionally insert a signature inside the infected hosts in order to avoid reinfection of the same files, which could cause their size to grow enormously.

The payload component, if present, defines the usually harmful intentions of the virus and the exact instructions to be executed on the infected host. Such actions may include deleting or corrupting files on the user's system or stealing information and performing more advanced and sophisticated attacks. The trigger mechanism on the other hand, specifies the conditions upon which the payload will be delivered such as for example at a specific time or date.

For example, one of the most damaging and destructive viruses ever to have appeared in the wild is the CIH virus also known as the "Chernobyl" virus which was first detected in 1998 in Taiwan[18]. According to Symantec's Security Response[19], the CIH virus infected around one million computers in Korea alone, causing more than 250 million US dollars economic damage. The virus was designed to attack 32-bit Windows executable files and was triggered to be activated for the first time on the 26th of April, 1999. Chernobyl carried two different injurious payloads. The first payload was responsible for corrupting the victim's hard disk by replacing all of its contents with random data causing the system to crash, while the second one tried to permanently damage the computer by attacking the Flash BIOS and altering the stored data.

Viruses usually spread through removable storage media, shared folders, emails and unreliable internet downloads [15]. Moreover, if the infected file resides somewhere on a server, the virus, with appropriate human interaction, will most likely propagate throughout the network, thus infecting more computers [23].

Figure 2.3.1 presents Cohen's example of a simple virus and its three-partite structure in pseudo code [49].

```
program virus:=
{1234567;

subroutine infect-executable:=
 {loop:file = get-random-executable-file;
 if first-line-of-file = 1234567 then goto loop;
 prepend virus to file;
 }

subroutine do-damage:=
```

[18] http://www.techopedia.com/2/26178/security/the-most-devastating-computer-viruses

[19] http://www.symantec.com/security_response/writeup.jsp?docid=2000-122010-2655-99

```
 {whatever damage is to be done}

subroutine trigger-pulled:=
 {return true if some condition holds}

main-program:=
 {infect-executable;
 if trigger-pulled then do-damage;
 goto next;}

next:}
```

Figure 2.3.1: A simple virus example by Dr. Cohen.

Worms. Spafford in 1989 defined a computer worm as "*a program that can run independently and can propagate a fully working version of itself to other machines*" [53]. Worms, in general, are fully functional stand-alone programs that can be executed autonomously and replicate by creating copies of themselves as they move from one system to another. Their basic infection strategy resides mostly on exploiting system and network vulnerabilities with minimum or no user interaction. Due to the scale and magnitude of the attacks that they can accomplish as well as their proliferation speed, they are quite popular among cybercriminals aiming at performing massive system infections.

Viruses and worms are similar in the sense that they both have self-replicating capabilities. The main difference between them is that a worm does not need to attach itself to another file in order to propagate [52]. According to Dr. Cohen, the basic distinction between the two is the absence of an infection mechanism in worms [49]. This of course does not mean that a worm doesn't have the ability to modify or infect other files but rather that it does not require the presence of another program or file in order to proliferate. Skoudis, Zeltser and Szor state that what explicitly distinguishes a worm is its network-based infection mechanisms [15, 37]. Skoudis specifically denotes that "*if it doesn't spread across the network, it just isn't a worm*", while Szor further classifies worms as a subclass of viruses.

The basic structure of a worm is anatomically similar to that of a virus, in the sense that it also includes specific infection, trigger and payload mechanisms. Skoudis and Zeltser delineate the structural components of a worm in analogy to a missile, referencing to worms as weapons of war [15].

Table 2.3.2 presents and correlates the main component elements of a worm along with a small description of their functionality and mechanisms, as they are discussed in [15, 37, 57, 67, 68].

Table 2.3.2. Worm Component Elements

Worm Component	Functionality	Techniques / Actions / Mechanisms
Warhead	Consists of exploits that explore possible system vulnerabilities in order to gain access on a target machine.	Buffer Overflow, File Sharing, E-mail readers / servers, System and Network Misconfigurations / Flaws
Propagation Engine	Specifies the propagation methods necessary for the worm crawl and copy itself on the target, once the door is open.	Peer to Peer and File Sharing networks, File Transfer mechanisms (FTP, TFTP, HTTP, SMB), Instant Messaging, IRC channels, Web Services, owned SMTP engines, pre-installed malware.
Target Selection Algorithm	Defines how the worm searches for additional targets to attack.	IP Scanning/Generating, E-mail harvesting (address books, inbox, IE cash, personal directories, Google searches), Network shares / Neighbors, DNS queries, Hosts Lists, Service Discovery, OS Fingerprinting, Pre-configured Hit Lists, File System Traversal
Scanning Engine	Scans the identified targets to determine whether they can be also exploited by the warhead and if so, replicates and repeats the process on the new victim.	Sends packets to the addresses selected by the target algorithm to determine potential penetration opportunities
Payload	Any additional action to be executed on the compromised system.	Recruit Bots, Install Trojans, DDoS agents, spam servers, viruses, rootkits, spyware, backdoors, key loggers etc.

A worm's payload component might not always be present. In such cases the worm's ultimate goal is to quickly proliferate to as many machines as possible and the consequent damages are mostly due to the massive consumption of system and network resources [15]. The existence and nature of a payload mechanism on the other hand, might also indicate the attacker's motives and incentives [67]. Modern malware payloads, like for example the Storm worm's payload, demonstrate a multipartite nature and may include diverse mechanisms to be executed on the target machines.

Figure 2.3.2 displays the anatomy of the Bugbear.B worm, as it is presented by Skoudis and Zeltser [15]. The Bugbear.B worm was detected in 2003 and is identified as a new form of Combo malware which embraces multiple malware characteristics concurrently. Bugbear.B uses a combination of common worm propagation mechanisms to spread and its payload includes a mixture of attack techniques that blend together virus and backdoor features as well as polymorphic and antivirus disabling mechanisms.

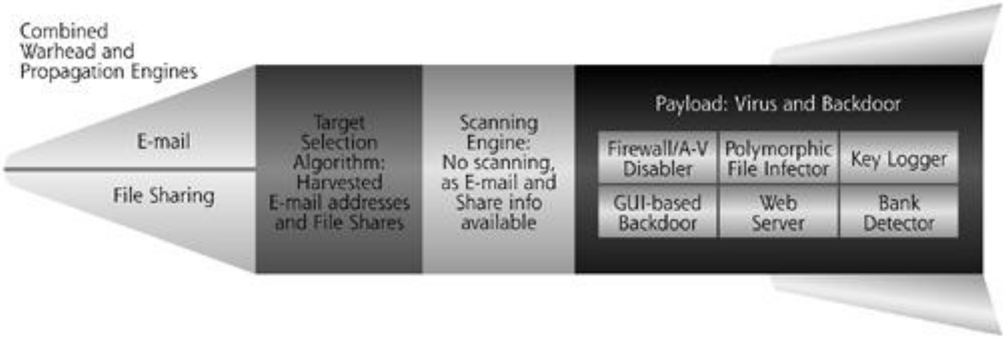

Combined Warhead and Propagation Engines
E-mail
File Sharing
Target Selection Algorithm: Harvested E-mail addresses and File Shares
Scanning Engine: No scanning, as E-mail and Share info available
Payload: Virus and Backdoor
Firewall/A-V Disabler
Polymorphic File Infector
Key Logger
GUI-based Backdoor
Web Server
Bank Detector

Figure 2.3.2: Anatomy of the Bugbear.B worm.

Such blended threats [37] have the ability to combine multiple infection strategies concurrently and are increasingly being utilized by modern malware, like the Conflicker worm, to augment their infection routines.

Worms can be generally classified into two major categories [52]:

- *Host computer worms*. Worms that run integrally on the victim's computer and use the network to simply replicate on other machines. *Rabbits* are host worms that terminate their execution on the previous machine after jumping on the next victim.
- *Network worms*. Worms that are divided into multiple segments which are then interspersed throughout the network. Each segment is executed on different computers and their overall communication takes place through the network.

Octopuses are network worms that contain one basic segment responsible for the management and coordination of all the other worm parts.

Based on their launching mechanisms, Harley, Slade, and Gattiker [52] additionally categorize worms to:

- *Self-launching worms* that can self-replicate without any human intervention.
- *User-launched worms* that employ social engineering techniques to trick users into executing their code.
- *Hybrid-launch worms* that combine both aforementioned techniques.

Even though recent attacks like the one performed by the ultimate cyber weapon Stuxnet have introduced a new hybrid, mixed type of worms that can additionally propagate through other means such as removable usb devices, most worm instances are network-based and can replicate by exploiting network and system vulnerabilities. Based on their propagation and payload mechanisms, worms can be further classified into [68]:

- Email Worms
- Instant Messaging Worms
- IRC Worms
- Internet Worms
- File-sharing Network Worms
- Peer to Peer Network Worms
- PDF Worms

Trojan Horses. Skoudis and Zeltser [15] define a Trojan horse as *"a program that appears to have some useful or benign purpose, but really masks some hidden malicious functionality".* Trojan horses in general are programs that disguise their actual nature and present themselves as innocent software programs pretending to perform something other than their true functionality. To further confuse the victims, some Trojans might actually execute what they are claiming to deliver in conjunction with the undesirable results. Harley, Slade, and Gattiker [52] describe Trojans as "*programs that claim to do something useful or desirable, and may do so, but also perform actions that the victim wouldn't expect or want*" and denote that what distinguishes them from viruses and worms is the lack of self-replicating properties and the absence of proprietary infection mechanisms.

Trojan makers try to deceive the victims and conceal their malicious programs behind common names of system processes such as “init” in Unix systems or “iexplore” in Windows. Several techniques might be engaged to hide executable suffixes and wrapper tools are used that merge the illicit code with other legitimate programs so as to be executed alongside them [15].

According to Kaspersky Labs’ classification[20] as well as related research work [15, 52, 56], several different types of Trojan programs exist and can be categorized according to their functionality:

- *Backdoors.* Trojan Backdoors are programs that provide attackers privileged access on a compromised computer [52]. Backdoors in general are utilized by cyber criminals to gain remote access on the infected machine with the purpose of controlling its behavior, remotely executing commands, installing additional malware, exchanging files and data or even to monitor and control the victim’s GUI. This can be done by illegitimately using legitimate administrative tools like Netcat, Back Orifice and Virtual Network Computing (VNC) [15].
- *Droppers.* Trojan droppers (also known as infectors [52]) contain various additional malware components, such as worms, backdoors etc., and are designed to secretly install and execute them on the victim’s computer. These might include new malicious code instances or updated versions of previously installed ones.
- *Downloaders.* Upon the successful connection to a remote server they secretly download, install and execute malware on the victim’s computer. They are smaller than droppers and are often used to dynamically download updated malware code versions.
- *PSW Trojans.* Password Stealing Ware programs are specifically designed to steal personal information and passwords which are then communicated to the controller.
- *Spies.* Trojan spies monitor and record user actions which are then dispatched to the attacker.

[20] Kaspersky Labs, http://www.securelist.com/en/threats/detect/trojan-programs?behavior=19

- *Proxies.* Work as proxy servers providing internet access and at the same time hiding the attacker's identity. They are most often used for massively distributing spam e-mails.
- *Notifiers.* Report to the controller relevant information regarding the installation of malware and the infection status of the victim's machine.
- *Arcbombs.* Archive bombs are compressed files constructed to crash the system upon any un-packing attempt. When decompressed they flood the disk with random data significantly affecting the system's performance. They are most often used to disrupt anti-virus software and crash mail and file servers.
- *Clickers.* Trojan Clickers are programmed to redirect the victim's computer to various, usually predefined, internet resources such as specific web pages. This is usually done with the purpose of increasing a website's traffic for promoting on-line ad campaigns, to perform a Denial of Service (DoS) attacks against a specific target or to lead the unaware users to an already compromised website for further infection.
- *DDoS Agents.* These programs are installed on the targets to organize DoS attacks against a specific victim machine. Usually the attackers infect multiple computers concurrently which are then coordinated to massively attack another machine.

Bots. The invention of the IRC[21] technology in 1988 led to the development of the first IRC bot in 1989 [61]. Bots (short for robots) were initially developed as virtual robot users to perform several human related actions and to assist their owners with the management of their IRC connections. Bots were capable of acting on behalf of their owners, while they were busy with other activities, and could occupy the IRC channel preventing others from taking it over. Bots evolved into effective tools that helped users manage IRC channels and provided various additional services such as enabling shell user accounts on the IRC host allowing remote commands execution. Pretty soon cybercriminals started to utilize IRC bots and related technology to promote their malicious activities and launch massive attacks [65].

[21] Internet Relay Chat (IRC)

A botnet is a collection of computers (bot clients) whose actions are secretly controlled and coordinated by a botherder with the purpose of achieving a common goal [61]. The botherder directs the bot clients' actions through remote IRC communication from a command and control (C&C) server. The infected computers that are under the botherder's control are also known as zombies and allow the attacker to manipulate their activities without even logging into their systems. Since the illegal actions are performed by the infected bot clients, the attackers remain invisible and undetectable behind the IRC channel and might further complicate investigators by using obfuscation and multiple hop techniques to direct their commands. Modern botnets append an extra layer of complexity as they usually comprise of a collection of bot servers, managed by the botherder, that each controls a different group of zombie machines. If one communication path is destroyed the rest remain unaffected [61].

According to Schiller et al [61] a botclient's lifecycle begins by its exploitation. This can be achieved by various means such as deceiving the user into executing the infectious code through phishing and spam mailing attacks, taking advantage of system vulnerabilities and backdoors opened by other malware invasions or brute force attacks on user passwords. Upon infection the freshly recruited bot initiates communication with its C&C server to notify the botherder and receive any possible updates necessary. It then receives and executes appropriate software to disable any antivirus programs and conceals its presence using rootkits and other tools. At this point the bot starts receiving commands from its controller, downloads and executes various payloads and sends reports back to the botherder, a process which is repeated until its final abandonment and termination.

Bots can be utilized by the botherder to perform multiple actions [61, 64, 71] such as:

- Act as recruiting agents to enlist other botclients.
- Perform coordinated DDoS attacks.
- Collect any type of personal, financial, or system information including identities, credit card or social security numbers and banking credentials.
- Execute massive spamming or phishing attacks.
- Storing and distributing illegal digital material.
- Organize ransoming attacks like for example encrypting the victims data and then extorting them in return for the decryption key or performing DoS attacks and demand ransom to terminate them.

- Secretly install adware or deceive on-line advertising vendors and affiliate networks with iconic website visits and ad hits.

Botnets can be utilized to perform massive or more targeted attacks. The botherder has the ability to customize the range of the targeted hosts through a desired predefined list of IP addresses. More over botclients' behavior can be adapted accordingly based on each infected host's system variables and installed applications. Modern botnets engage new communications mechanisms such as peer to peer technologies or Fastflux and dynamic DNS services that further enhance their presence and hinder their detection. Fastflux DNS technology for example, which was used by the recent Storm Worm, constantly and repetitively changes the IP address of the C&C server while preserving the same Domain name. The use of such technologies encumbers bot servers' detection and containment, if not making it impossible [61].

Botnet technology is widely used by cybercriminals, as a major profit making opportunity, forming new underground crimeware business models [64]. According to Schiller et al [61] a spamming campaign alone can generate up to $750,000 profit per month. With the global range and scale of attacks that botnets can accomplish, the underground economic market is flourishing. According to a report by Kaspersky Labs related to the economics of botnets [63], botnets are prevalent in the underground market due to their low maintenance costs as well as the limited knowledge skills required for their management. Figure 2.3.3 is imprinted from Kaspersky's report [63] and illustrates how botnets can generate money for their creators.

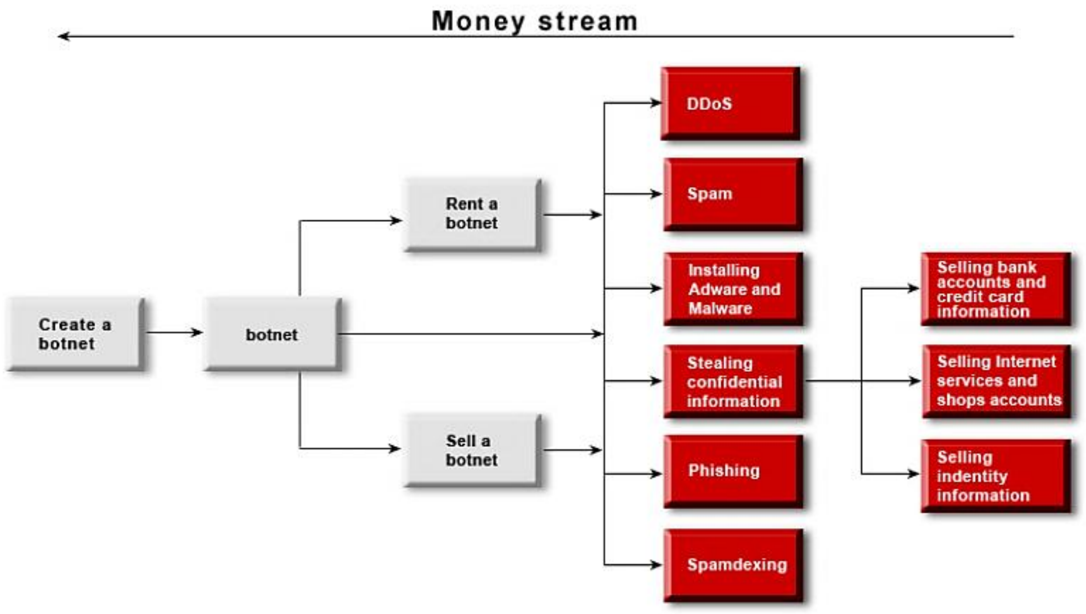

Money stream
Create a botnet
botnet
Rent a botnet
Sell a botnet
DDoS
Spam
Installing Adware and Malware
Stealing confidential information
Phishing
Spamdexing
Selling bank accounts and credit card information
Selling Internet services and shops accounts
Selling indentity information

Figure 2.3.3: The money making business of Botnets

Latest research studies reveal that cybercriminals use botnets as their main platform for organizing global cyber-attacks [69]. Schiller et al [61] denote that "*Today's bots are easy to customize, modular, adaptive, targetable, and stealthy*" and identify botnet technology as one of the major driving forces behind organized cybercrime with global impacts that can threaten not only large organizations but even entire nations.

Rootkits. Rootkits are programs specifically crafted to hide and conceal malicious activity by substituting running processes, network traffic or system and registry files [52]. They are used to avoid detection mechanisms and to prolong their presence on the infected machine.

Spyware. Spyware is a general term used to describe any type of software that gathers information from users' machines without their knowledge or consent [65]. The information collected might include personal or financial data, passwords, browsing habits and history, banking and credit card credentials etc. Advanced spyware might also interfere and modify system settings or consume various resources affecting the computer's performance.

Adware. Adware programs are secretly installed on the users' machines with the purpose of automatically downloading and launching various types of advertisements [65]. They most often create a web browsing profile of the victim which is then used to direct targeted pop-up advertisements based on the user's interests.

Key-loggers. Keystroke loggers are programs that monitor and record every typing action on a user's keyboard [65]. Keystrokes are logged and saved on the victim's disk, which are then later communicated to the attacker. Keyloggers are commonly used by cybercriminals that want to retrieve confidential information to be used for illegal purposes and financial gain.

Table 2.3 presents a generic taxonomy of the aforementioned malware types based on their propagation mechanisms and host requirement.

Table 2.3: Malware taxonomy

	Host Requirement	
Propagation method	*Host Required*	*Independent*
Self-Replicating	Viruses	Worms
Non-Replicating	Trojan Horses, Rootkits	Keyloggers, Spyware, Adware

2.6 Types of Attacks

Cybercriminals engage malware in order to serve their malevolent goals and have the ability to manifest various types of massive or targeted cyber-attacks such as prohibiting access to information systems and services, performing espionage, stealing confidential or identity information, conducting on-line banking fraud and even extorting potential victims for financial profit [71]. The following paragraphs provide a description of some of the most common types of cyber-attacks.

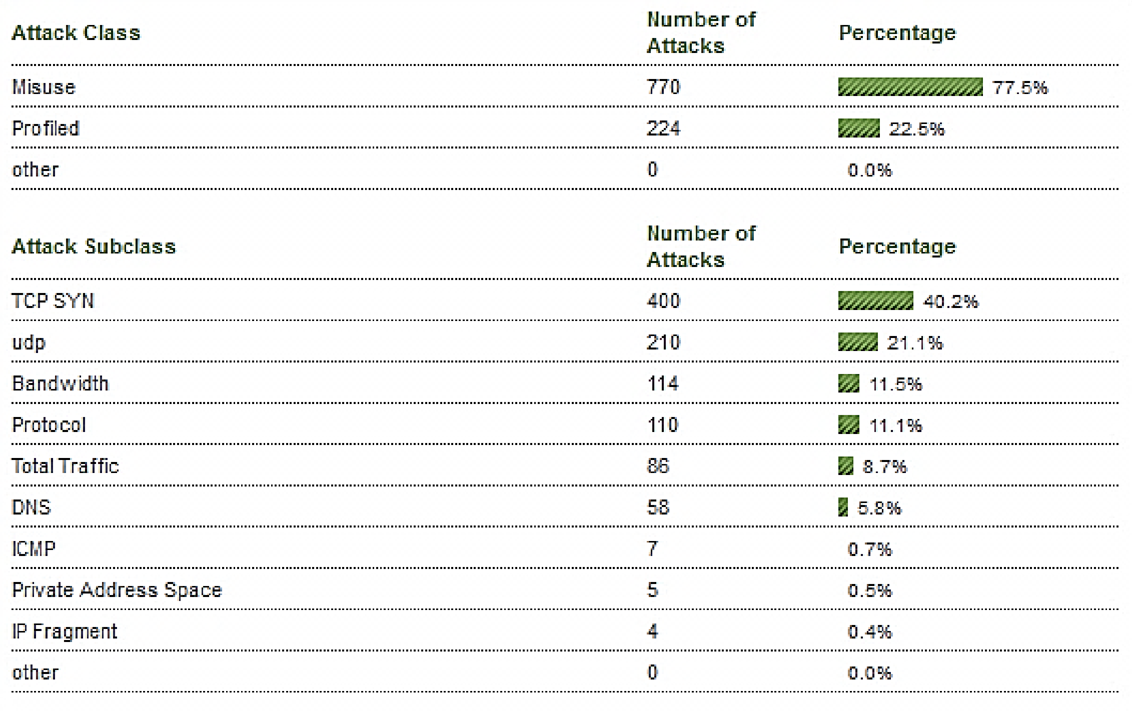
DoS attacks. The main goal of a Denial of Service (DoS) attack is to make services and applications inaccessible to end users [65]. DoS attacks do not involve bypassing security mechanisms but rather focus on tying up network and system resources leading to diminished network connectivity and unavailable services and applications [52]. This is usually achieved either by consuming network bandwidth or by issuing an excessive number of connection requests. Examples of DoS attacks include flooding the network with large volumes of data or massively dispatching e-mails degrading network connectivity and consuming the system’s disk space. Another form of DoS attack involves clogging the system’s processing power for example by performing login attempts forcing the computer to authenticate the requests. A variant type of attack could relate to forbidding user access on a system by repetitively entering invalid passwords until the user is locked out [65].

Two of the most common types of DoS employed technologies are TCP SYN Flood and UDP Flood attacks [61]. TCP SYN flood attacks exploit the TCP handshake process that most network applications rely on. The attacker sends a large number of SYN requests to a server, without responding to the SYN-ACK acknowledgement messages of the receiver. This is achieved either by simply not replying to the messages or by spoofing (forging) the sender’s IP address. The server on the other hand binds resources and time while waiting for the response. A large amount of SYN messages could set the receiver unresponsive to other legitimate TCP requests. UDP flooding on the other hand, entails directing a massive amount of trivial UDP packets to randomly selected ports, blocking out regular network traffic by consuming the system’s processing power and bandwidth.

To further enhance their attack power and destructive outcomes attackers employ bot networks and launch *Distributed Denial of Service* (DDoS) attacks [52, 65]. Thousands of infected zombie machines are controlled and coordinated by the botherder and through various simple commands can be accordingly instructed so as to serve the attacker’s goals. The bot master can direct multiple geographically scattered bots to concurrently conduct DoS attacks against a specific target. Through such massively synchronized DDoS attacks, cybercriminals are able to significantly affect and damage not only powerful companies and organizations but also threaten entire countries and nations [65].

Arbor Networks, a major DDoS research and response group, monitors global DDoS attacks on a 24 hour basis and provides insights and intelligence analysis on worldwide DDoS threats. Figure 2.3.4 presents the amount and type of global DDoS attacks as they were recorded by Arbor's Threat Level Analysis System (ATLAS) during a very recent and specific 24 hour period[22], with TCP SYN and UDP flooding techniques counting to more than 60% of the overall number of realized DDoS attacks.

[22] http://atlas.arbor.net/summary/dos, Accessed at 16-10-2012. DDoS summary refers to attacks realized between 15-10-2012 and 16-10-2012.



Attack Class	Number of Attacks	Percentage
Misuse	770	77.5%
Profiled	224	22.5%
other	0	0.0%

Attack Subclass	Number of Attacks	Percentage
TCP SYN	400	40.2%
udp	210	21.1%
Bandwidth	114	11.5%
Protocol	110	11.1%
Total Traffic	86	8.7%
DNS	58	5.8%
ICMP	7	0.7%
Private Address Space	5	0.5%
IP Fragment	4	0.4%
other	0	0.0%

Figure 2.3.4: Global Distributed Denial of Service (DDoS) attacks Summary.

The reasons behind a DoS attack vary and may involve extorting victims for profit, overpowering and unlawfully gaining competitive advantage against rivals by disintegrating competitor services subsequently causing substantial financial losses or even organizing daunting terrorizing cyber-attacks [63, 64, 65]. For example in early 2009 a coordinated DDoS attack was realized against godaddy.com, a large ISP company, leading to thousands of disrupted hosted client websites that remained of-line for about 24 hours [63]. The motivations are yet to be clarified but speculations involve either a rival company's attack or a blackmailing attempt.

Cybercriminals go even further, indiscriminately advertising pay-per-hour or pay-per-day DDOS services[23]. The ease, by which DDoS attacks can be realized, as well as the massiveness of attack sources and the magnitude of the resulting outcomes, marks DDoS as one of the most intimidating forms of cybercrime [65].

Information and Identity Theft. Identity (ID) theft relates to illegally acquiring, communicating or abusing personal information with the purpose of committing fraudulent activities or other related crimes [71]. Attackers usually employ social engineering

[23] http://ddos.arbornetworks.com/2012/07/ddos-attacks-targeting-traditional-telecom-systems/

techniques, such as distributing phishing e-mails, to deceptively persuade victims into disclosing private information to a presumably reliable source. Cybercriminals engage botnets to send out spam e-mails that might contain links to deceitful websites that appear as legitimate companies to deviously retrieve information. Other types of spam e-mails may embed malicious code or lead to already compromised sites so that malware will be automatically installed on the victims' machines enabling the collection of confidential and private information.

Figure 2.3.5 illustrates how an attacker can perform an on-line identity theft attack, utilizing malware, botnets and social engineering techniques to eventually extract the desired information from the victims and their systems [71].

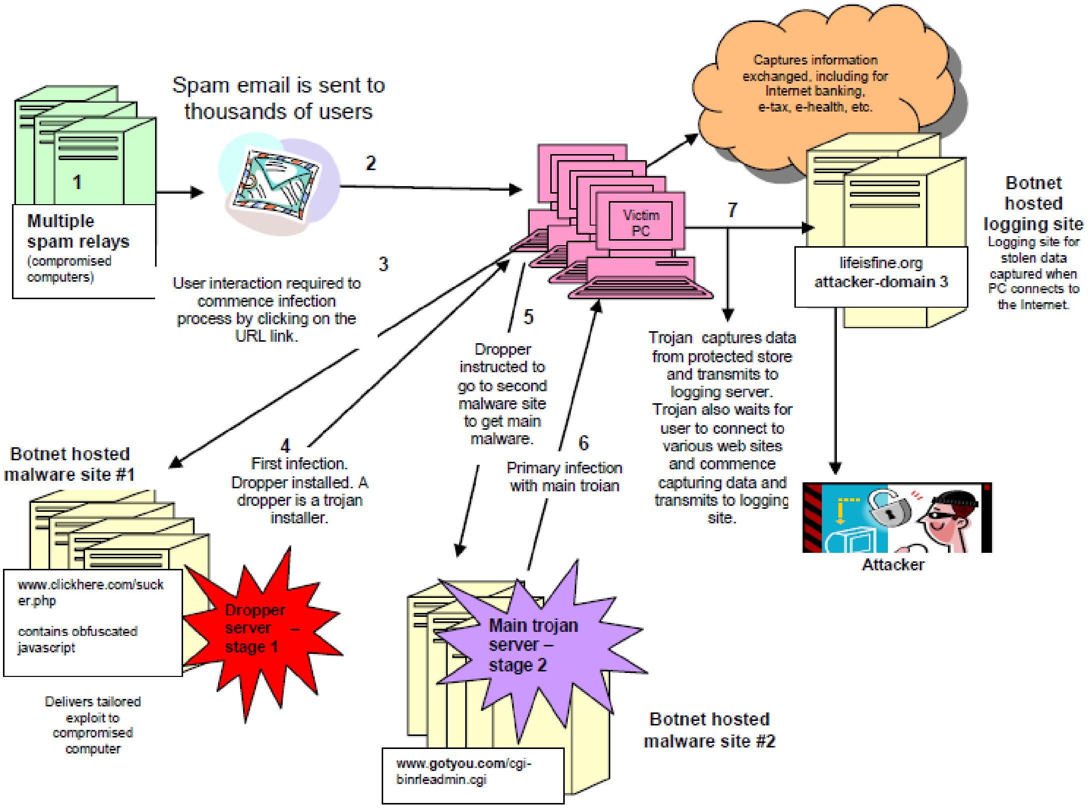

Spam email is sent to thousands of users
Captures information exchanged, including for Internet banking, e-tax, e-health, etc.
1
Multiple spam relays
(compromised computers)
2
Victim PC
7
Botnet hosted logging site
Logging site for stolen data captured when PC connects to the Internet.
lifeisfine.org
attacker-domain 3
3
User interaction required to commence infection process by clicking on the URL link.
5
Dropper instructed to go to second malware site to get main malware.
Trojan captures data from protected store and transmits to logging server. Trojan also waits for user to connect to various web sites and commence capturing data and transmits to logging site.
Botnet hosted malware site #1
4
First infection. Dropper installed. A dropper is a trojan installer.
6
Primary infection with main troian
Attacker
www.clickhere.com/sucker.php
contains obfuscated javascript
Dropper server – stage 1
Delivers tailored exploit to compromised computer
Main trojan server – stage 2
Botnet hosted malware site #2
www.gotyou.com/cgi-binrleadmin.cgi

Figure 2.3.5: Example of an organized Identity Theft attack using malware.

Espionage. Malware can also be engaged by criminals to illegally penetrate on various systems and perform political, industrial or even nation-wide espionage attacks [71]. Malicious cyber-spies use malware to gather confidential data by breaching the private or public sector's security infrastructure. Espionage attempts are also often found among rival organizations that seek to attain information regarding competitors' operations.

On-line Banking Fraud. Information collected from unaware victims as well as specific malware creations are most often related to illegal withdrawals or fraudulent conveyance of funds from bank accounts [71]. The very recent "High-Roller" attacks[24] demonstrated how malware can be utilized by on-line robbers to perform targeted, extremely automated and sophisticated banking cyber-thefts. The attackers combined and specifically customized the SpyEye, Zeus and Ice 9 malware toolkits targeting only high balance bank accounts. They infected the bank clients' computers and illegally transferred the funds while the users were prompted with waiting messages during their on-line transactions.

Targeted attacks. Targeted attacks involve specially crafted malware instances designed to specifically target a particular organization or company [22]. Matrosov et al. [70] further describe an additional class of targeted attacks that does not focus at a specific company but target certain types of software and IT related infrastructure such as for example a malware attacking banking software or SCADA systems like the recent Stuxnet worm.

Targeted attacks are amongst the hardest to detect or analyze and pose a major security threat for organizations and large corporations as such malware attacks are extremely sophisticated and may go unnoticed for a large period of time magnifying the resulting damaging effects [22, 23].

Infection Vectors.

A malware's ability to penetrate and invade target systems is reinforced and facilitated by various infection vectors that are exploited by malicious authors in order to augment their chances of infiltrating and polluting unaware users' machines. Some of the most common propagation mechanisms, as they are discussed by Egele et al. [23], as well as exploitation attempts on new IPv6-based networks [46] are described in the following paragraphs.

Exploiting Network Services Vulnerabilities is a famous propagation method most often used by worms. An employable pathway is often found in services that are provided

[24] http://www.enisa.europa.eu/media/press-releases/eu-cyber-security-agency-enisa-201chigh-roller201d-online-bank-robberies-reveal-security-gaps

over the network, running on a server, to a large number of clients sharing the same resources. If a vulnerability of such a service is discovered then the malicious code could be executed on the server, subsequently allowing the malware to automatically spread over the connected systems.

Drive-by Downloads typically concern unintentionally downloading malicious software from the internet. Malicious attackers search for Web browser and plug-ins vulnerabilities with the intent of installing and executing the maleficent code on the victim's machine. In order for such an attack to be realized, the user must first visit a malicious or tampered web site. Assailants try to manipulate users by deploying spam emails enclosing links to malicious sites, displaying misleading pop-up windows, tricking search engines' ranking mechanisms or tainting already existing vulnerable pages, not necessarily part of popular or high traffic sites.

For example, a drive-by download attack, which exploited an ActiveX's vulnerability with the purpose of installing a Trojan on the users' machines, took place recently at a pizza online ordering website where the attackers used SQL injection as a method for inserting an iFrame into the website's code, referencing to a malicious site[25]. A variant, yet similar *drive-by cash* attack, in which the malicious file is not downloaded and installed on the victim's computer but executed through the browser's cash, exploited a security hole in Adobe's Flash 0-day player and infected Amnesty's International Human Rights web site by inserting spyware camouflaged as a javascript file[26].

IFrames, a browser feature used to insert the content of a webpage inside a part of another page, is one of the most popular methods for infecting legitimate websites[27]. Cybercriminals exploit this vulnerability and inject invisible iFrames, usually containing encrypted or packed executable code which is placed in a remote site, into existing webpages. Typically the inserted code is a downloader that contains a simple redirect command pointing to another IP address. When a user lands on an iFramed page the downloader is executed and the browser is instructed to visit the malicious IP which in turn, for obfuscation reasons, may contain another downloader referencing to another

[25] http://www.h-online.com/security/features/CSI-Internet-Alarm-at-the-pizza-service-1019940.html

[26] http://blog.armorize.com/2011/04/newest-adobe-flash-0-day-used-in-new.html

[27] http://www.cio.com/article/135452/Death_by_iFrame?page=1&taxonomyId=3089

address. This process is often repeated many times until the malware is finally dropped into the user's machine.

Another popular method of deceiving users into visiting malicious websites is by taking advantage of high ranking search engine keywords most often involving famous celebrities. Attackers monitor the latest trends and hide malicious software behind webpages that pretend to display relevant information, pictures and videos regarding favorite superstars. According to McAfee's recent study, Emma Watson was declared as the most dangerous cyber celebrity of 2012[28]. Their research reveals that there is a 12.6% possibility of visiting a malware flooded website when searching for pictures and videos of Emma Watson. Downloading any kind of material from such a site will also result into receiving all sorts of spyware, viruses, adware etc. along with the desired content.

Social Engineering forms another major infection vector and refers to all possible tricks and ingenious appealing means that attackers engage in order to trap and allure oblivious users into compromising their own machines. This could involve conducting spamming and phishing campaigns or deceptively uploading malicious links on social networking platforms, blogs and forums. According to Microsoft's Intelligence Report [31], which involves information collected in the first half of 2011, approximately 44.8 % of the noted propagation mechanisms required human interaction for the malware to spread while only 6% of malware relied on exploiting software vulnerabilities.

IPv6 attacks. Recent studies have shown that the new internet protocol is also susceptive to potential internet threats and faces some serious flaws and security concerns that make IPv6 based networks vulnerable to cyber-attacks [45, 46, 47, 48]. Researchers have demonstrated that IP fragmentation attacks are also possible in IPv6 networks and may result to firewalls evasion, OS fingerprinting, Intrusion Detection Systems (IDS) insertion/evasion as well as remote code execution [46]. Furthermore the absence of sufficient malware defense mechanisms in mobile and networked devices, such as iPhones, Android-based gadgets, iPads etc., enables the opportunity for such devices to be recruited as active botnet members or to be utilized as an infection vector, facilitating malware distribution and transcendent attacks on other interconnected networks [45].

Cybercriminals have become aware of these security weaknesses and have already started to target the new communications protocol and exploit its vulnerabilities. In fact,

[28] http://www.mcafee.com/us/about/news/2012/q3/20120910-01.aspx?cid=110907

according to Arbor Networks' 7th annual Worldwide Infrastructure Security Report[29] which was published in February 2012, the first ever DDoS attacks against IPv6 networks have already been realized and recorded for the first time in 2011. An example of such a DoS attack is the one performed by the *IPv6f**k* malware which uses *TCP SYN flooding* as a resource depletion method in order to immobilize network services[30]. Other types of attacks can also be realized in IPv6 such as address spoofing and redirecting network traffic through ICMPv6 redirects.

Even though the attacks on IPv6 based networks still remain rare, they are considered to be an emerging threat as the traffic volume in such networks is growing rapidly and the adoption of the new network layer protocol is being accelerated by the massive expansion of the mobile and other networked electronic devices market.

Modern malware samples may exhibit multipartite behavior [52], meaning that they might engage multiple infection strategies (for example a virus infecting both the boot sector and executable files) in order to accomplish their malevolent purposes on variant platforms. Blended threats additionally might use various infection vectors, exploiting multiple vulnerabilities, to propagate and spread [37]. For example an attacker could send a spam email luring the victim to visit a legitimate but iframed website. As soon as the user lands on the tampered page the malware is installed on the target computer and upon execution a specific file could be dropped which in turn will download additional malicious software such as Trojans, Spyware, Key-loggers etc.

2.7 Malware Forensics

In-depth malware analysis techniques are increasingly being utilized within forensic investigative procedures [28, 42]. By uncovering malware functionality and examining possible traces and patterns left behind on the infected host, digital investigators can retrieve substantial intrinsic information that enable the effectiveness of investigative processes as well as the successful discovery of advanced malicious intrusions and cybercrimes. The significance of malware analysis within digital investigations combined with the continuous evolvement, sophistication and complexity of malware code has led to the development of more complete and formal malware forensics frameworks [21].

[29] http://www.arbornetworks.com/report

[30] http://zvelo.com/blog/ipv6-malware-virus-worm-dos-attacks-tunneling-examples-samples

Modern malware forensics methodologies incorporate both malware analysis techniques and forensic analysis tools to produce comprehensive and reliable analysis results that can substantiate investigative procedures and provide usable and valid prosecution evidence.

Cyber criminals nowadays are constantly trying to engineer new anti-forensic mechanisms thus hindering a forensics' analyst job and thwarting the analysis process [23, 32, 39]. Anti-forensics is a generic term used to describe techniques employed to escape both forensic detection and forensic analysis procedures and can be generally achieved either by destroying residues of data or by hiding or not storing it at all on the disk [32]. The reason being that concealing network traffic and hiding file system traces left on the compromised computer obstructs digital investigators from unveiling the malware's behavior and discovering malicious intrusions as well as eliminates possible evidence that may lead to the intruder's identity [21]. Moreover, the longer the malware remains undetected or hides its malicious profile the greater the chances of succeeding and profiting by achieving its goals, consequently causing the maximum possible damage.

Malware authors manifest sophisticated mechanisms to impede reverse engineering processes as well as both static and dynamic analysis procedures. The following sections highlight and describe some of the most commonly employed anti analysis techniques as they are presented and discussed in relevant research papers [20, 23, 73].

Obfuscation and Packing Mechanisms. Malware creators use obfuscation techniques in an attempt to avoid detection and impede the static analysis of the program by modifying the malicious code and hiding its actual intentions. This means that the code under analysis is not the actual code being executed [23]. Obfuscation can be achieved through various techniques such as for example dead code injection, code permutation and instruction substitution [33, 66].

Dead code insertion injects trash code into the program leaving its semantics unchanged. Code permutation or transposition changes the order of the instructions either by swapping the independent ones or by arbitrarily rearranging them and adding unconditional branches to preserve the initial control-flow. The first method aims at creating a binary representation with a different instruction stream than the one included in the malware's signature that Anti-Virus vendors use for detection, while the second one focuses on differentiating the binary's instruction order from the execution order. Instruction substitution involves replacing sequences of instructions with other semantically equivalent order sets usually drawn from a predefined instruction dictionary.

Christodorescu et al. [33] present a clarifying example of how these techniques can be applied in practice by modifying the detection signature of the Chernobyl (CIH) virus. Figure 2.5.1 shows the original IA-32 code fragment, generated from the hexadecimal sequence "E800 0000 005B 8D4B 4251 5050 0F01 4C24 FE5B 83C3 1CFA 8B2B" which is used by Anti-virus software in order to detect the Chernobyl virus, as well as the obfuscated code produced by the aforementioned techniques. The newly generated code retains the same functionality but corresponds to a different signature pattern than the original one.

Table 2.5.1: Example of Obfuscation Techniques

	Obfuscation Technique			
Original Code	*Dead Code Insertion*		*Code Transposition*	*Instruction Substitution*
call 0h	*call 0h*		*call 0h*	*call 0h*
pop ebx	*pop ebx*		*pop ebx*	*pop ebx*
lea ecx, [ebx+42h]	*lea ecx, [ebx+42h]*		*jmp S2*	*lea ecx, [ebx+42h]*
push ecx	*nop*	*S3:*	*push eax*	*sub esp, 03h*
push eax	*nop*		*push eax*	*sidt [esp - 02h]*
push eax	*push ecx*		*sidt [esp - 02h]*	*add [esp], 1Ch*
sidt [esp - 02h]	*push eax*		*jmp S4*	*mov ebx, [esp]*
pop ebx	*inc eax*		*add ebx, 1Ch*	*inc esp*
add ebx, 1Ch	*push eax*		*jmp S6*	*cli*
cli	*dec [esp - 0h]*	*S2:*	*lea ecx, [ebx+42h]*	*mov ebp, [ebx]*
mov ebp, [ebx]	*dec eax*		*push ecx*	
	sidt [esp - 02h]		*jmp S3*	
	pop ebx	*S4:*	*pop ebx*	
	add ebx, 1Ch		*cli*	
	cli		*jmp S5*	
	mov ebp, [ebx]	*S5:*	*mov ebp, [ebx]*	

Such obfuscation mechanisms can significantly obstruct manual static analysis approaches. Tools like "SAFE" [33], a static malware analyzer introduced by the aforementioned authors, have the ability to detect some of the most common obfuscation techniques. SAFE creates an abstract representation of the malware's code and executable and detects malicious patterns, generated from the abstracted code, inside the generalized version of the executable.

Other common obfuscation techniques, usually employed by self-modifying programs, are polymorphism, used to avoid signature matching antivirus software, and metamorphism, employed to escape heuristic analysis techniques [33, 37]. A polymorphic malware has the ability to decrypt its contend during execution with randomly chosen keys while a metamorphic one can automatically mutate and recode itself every time it is proliferated and unpacked. Modern malware samples implicate more complex and advanced obfuscation mechanisms using emulation technologies that transform malicious binaries into randomly generated instruction sets which are interpreted by an enclosed binary emulator [72].

More recently Wenke Lee et al. [34] presented a malware obfuscation technique with the ability to cover the malware's trigger based behavior by automatically encrypting the parts of the code that are activated by a specific input value, using that same value to create the encryption key which is afterwards taken out of the program. This mechanism was devised to specifically obstruct analysis procedures that use multiple path exploration, symbolic or conditional execution processes for identifying trigger based malware behavior [13, 28]. Several other obfuscation mechanisms can also be found in [35, 36, 74].

Packer programs are used in order to transform an executable into a different form while preserving the same functionality [23]. The packer automatically obfuscates or encrypts the original code, prepending also an unpacker responsible for reverting the data to its initial form, resulting to a new executable. The entire unpacking procedure takes place solely in memory and is activated during loading.

Dynamic analysis is generally not affected either by obfuscation or by packing techniques [23]. The reason is that once the program is unpacked it will be executed and perform its malicious actions. On the contrary, static analysis that depends on binary examination may be significantly obstructed. To overcome this obstacle, the packed program needs first to be unpacked either automatically by using unpacker programs, such as for example Renovo [77], OmniUnpack [76] and PolyUnpack [75], or manually with the help of a debugger and other related tools [20].

Malware authors have also come up with procedures to obstruct dynamic analysis and advanced reverse engineering techniques [22]. Modern malware instances can detect the usage of analysis tools and either modify their performance or remain dormant during analysis processes [23, 39]. Table 2.5.2 correlates and summarizes some of the most commonly employed anti-forensic techniques that target dynamic and reverse engineering methodologies as they are presented and discussed by Brand et al. [20, 73], Egele et al [23] and Sikorski et al. [22] in their related research work.

Table 2.5.2: Summary of Anti Forensic techniques employed by malware

Technique	Description
Detection of Analysis Environments	Malware has the ability to identify the use of an instrumented virtualized or emulated analysis environment through: • Hardware fingerprints in virtual machines. • The existence of external monitoring applications like debuggers or registry tools. • Behavioral differences between emulated and real hardware systems such as CPU bugs or timing variances. • Artifacts that exist in a monitored execution environment like status flags of a debugger etc.
Detection of On-line Analysis	Various techniques allow malware to identify whether it is executed in an on-line analysis tool like Anubis.
Anti Tools	Modifies its behavior upon the detection of certain analysis tools.
Anti Debuggers	If a debugger is detected malware can manipulate the execution flow to deceive analysis results.
Anti Disassembly	Utilizes the disassemblers' functionality to generate incorrect disassembly results.
Logic Bombs	Instead of attempting to detect the analysis environment malware can devise logic bombs to conceal malicious activities. A

	logic bomb unveils its behavior only under certain conditions such as on a specific date or user input.
Analysis performance	To enhance an automated analysis system’s throughput (amount of analyzed malware instances per time unit) and to allocate time accordingly so as to manage all necessary analysis tasks, a usual approach is to terminate the execution after a predefined timeout period. The behavior of malware samples that deliberately delay the generation of relevant processes might not be captured.
Rootkits	Malware uses rootkits to deceptively conceal malicious processes.

To countermeasure and mitigate anti forensic mechanisms, several methodologies and frameworks have been proposed [25, 40, 81, 82]. Aquilina et al. in 2008 recommended a malware forensics methodology, in order to address the problems of forensic analysis in a more holistic manner, which integrates the forensic analysis procedure into a wider investigative and forensic framework [21]. The proposed forensic methodology is divided into five subsequent investigative phases:

- Conservation and inspection of volatile data using forensic tools.
- Memory analysis.
- Investigation of hard drives using forensic analysis.
- Static analysis.
- Dynamic analysis.

2.8 Malware Analysis

Malware analysis, in a broadest sense, refers to all the necessary techniques and procedures that analysts employ in order to dissect, to examine and to completely unfold a malware’s inner structure in an effort to unveil all possible aspects of its malicious behavior and functionality. Kris Kendal in 2007 defined malware analysis as “*The action of taking malware apart to study it*” [43], while Moser et al. refer to malware analysis as “*the process of determining the behavior and purpose of a given malware sample*” [13].

Before digging further into specific analysis techniques and methodologies it is important to understand the significance of malware analysis and what it tries accomplish.

Security analysts constantly have to face and protect their organization from new emerging cyber threats involving the loss of private and confidential corporate information, industrial espionage, financial theft or any other possible attacks related to cybercrime. One of the most recent examples of cyber espionage broke out in Peru and some surrounding countries in 2012[31]. A worm called ACAD/Medre.A injected an infected AutoCAD template to a number of companies that used the respective software. Every time a user opened a drawing, a copy was dispatched to more the 40 mail boxes hosted at two different Chinese ISPs. According to ESET, who first detected the threat, approximately 100,000 drawings were stolen before it was finally contained.

A security analyst, facing such an attack, must take some immediate actions in order to minimize the company's loses and exposure, and answer some significant questions for preventing a similar intrusion in the future. By conducting malware analysis, the analyst will be able to understand the purpose of the malware, what it does exactly, how long it remained in the system, what exactly was stolen, how it was stolen, who attempted to steal it, how to contain it, and how to defend against future similar attacks. Consequently, by realizing how a specific malware operates, the analyst has the ability to assess the damages that were caused, to identify the exploited vulnerability and enhance the organization's defenses [43].

As Distler outlines "*The goal of malware analysis is to gain an understanding of how a specific piece of malware functions so that defenses can be built to protect an organization's network*" [14]. Sikorski and Honig [22] further denote that the goal of malware analysis is to deliver significant information so as to properly address a malicious intrusion. This involves identifying all possible compromised systems and files inside the company's network, precisely determining the malware's functionality and understanding how to estimate and confine the consequent damages.

Anti-virus vendors utilize malware analysis techniques in order to identify weather a suspicious sample is malicious. Malware analysis provides insights on the exact behavior of a sample and the intentions of malware creators. Thus appropriate detection and mitigation mechanisms can be developed to address new emerging threats [23].

[31] http://www.pcworld.com/article/258245/malware_gets_snoopy.html

Two essential techniques of malware analysis are available, namely static and dynamic analysis. Static analysis focuses on examining the malicious code without executing it, while dynamic analysis monitors the malware's actions during execution [22].

Static analysis is often hindered by malicious authors who apply obfuscation and packing mechanisms to prevent the inspection of the original code. This means that the code under investigation might not be the code that is actually executed [13]. On the other hand dynamic analysis might not provide inclusive results as malevolent performance might be stalled or hidden upon the detection of a simulated execution environment [23].

2.8.1 Static Analysis

Static analysis refers to the process of analyzing programs without having to execute them [22]. Static analysis enables the extraction of important information on the code's functionality and structure from an executable using a variety of available tools. The malicious code is analyzed through call graphs, strings identification, corresponding assembly instructions, control or data flow graphs, function and library calls and various other code artifacts that can be possibly rebuilt [19]. Such techniques are applicable on various code representations such as for example the binary equivalent of the program [23]. Various static analysis approaches have been demonstrated [33, 78], that enable and utilize static analysis methodologies.

The main advantage of static analysis is that it is relatively faster than dynamic analysis methodologies and it provides the ability of examining all possible aspects of the malicious code thus possibly uncovering the complete malware's behavior [13]. On the other hand a basic consideration concerning static analysis is that the inquiries involving the properties and functionalities of a given malware are often undefined which forces analysts to work with approximations regarding the solution of a specific problem [12]. This approach however, can be proven ineffective when examining malware, as it can be directly constructed by the attacker to deliberately thwart analysis procedures.

The static analysis process might be significantly obstructed through various obfuscation and packing mechanisms that are widely employed by self-modifying malware instances [23]. Such manifestations might hinder static code analysis approaches as well as lead tools like disassemblers to generate ambiguous assembly instructions that do not correspond to the actual executed ones. Moser et al. [74] introduced an obfuscation technique that uses opaque constants to overcast the control flow of the program and demonstrated that even modern semantics-based detection mechanisms can be eluded. Moreover, since the malware's source code is not available beforehand, the range of static techniques is limited to those the extract information from the malware's binary representation [23]. Malicious code that relies on dynamically defined variables and values (such as the system's time or date) can further intensify and hinder static analysis processes.

Recent efforts, like the development of the Eureka framework [78], try to improve the efficiency of static analysis techniques and to mitigate relevant limitations. However, the increasing complexity, structure plurality and sophistication of malware code, enhances the necessity for the development of more resilient and reliable static analysis procedures [23].

2.8.2 Dynamic Analysis

Dynamic analysis refers to the process of executing the malware in order to monitor, examine and analyze the performed actions [23]. Different types of dynamic analysis approaches are available and can be utilized to gain insights and retrieve essential information on the malware's behavior and functionality [38].

Behavior-based malware analysis monitors the actions of a given malware sample during run time. The malicious code is executed in an instrumented and controlled analysis environment, and its behavior and interactions with the host system are observed and recorded. Appropriate tools provide the opportunity to examine registry and file modifications, network traffic and packets, created or deleted files, generated processes, loaded DLLs, API calls, memory and disk usage as well as multiple other related information. Various frameworks have been introduced that implement behavior-based malware analysis [26, 79, 80]. This "black box" type of approach reveals significant information on the malware's exhibited behavior but fails to expose the program's inner structure and logic.

Comparing system's snapshots involves executing the malware in a simulated environment and identifying differences between snapshots that capture the system's state before and after execution. Implementing and applying such a technique as a stand-alone process can be done with relative ease however the obtained analysis results are quite coarse-grained in nature as any intermediate actions, such as for example the creation and deletion of files during execution, will be missed. Distler [14] incorporates this technique in a combined malware analysis approach as a supplementary process to enhance the resulting analysis outcomes.

Dynamic code analysis observes the program's activities while it is executed, usually with the use of specialized tools like debuggers. Debuggers provide control over the program's execution which can be intercepted, restricted and modified. Analysts have the ability to monitor memory and registry values, function calls and passing arguments as the code runs. In contrast to disassemblers that generate a static assembly representation of the code exactly before its execution, debuggers provide a dynamic insight on the malware's behavior. Modern dynamic analysis approaches [22] employ debuggers as the means to better understanding the internal structure of malware and enrich the attained analysis information.

As opposed to static analysis techniques, dynamic analysis has the advantage of analyzing the actual executed instructions [12]. To this end, dynamic analysis is basically unaffected by anti-forensic techniques such as obfuscation or packing mechanisms, as the code's functionality will eventually be demonstrated during execution. However, the main drawback of dynamic analysis is its inherently non-exhaustive nature in the sense that only a single execution path is monitored per analysis attempt. This means that the analysis results may not be inclusive regarding the malware's complete behavioral characteristics. Towards this end, researchers have developed new approaches that seek to expose possible trigger-based malware behavior [13, 28]. Nonetheless analyzing and extracting all possible behavioral characteristics is still a matter of ongoing research.

Moreover malware authors have developed various anti-forensic mechanisms that allow the detection of an instrumented analysis environment such as an emulator or a virtual machine as well as the use of automated on-line or locally installed analysis sandboxes [23]. If such a tool is identified, malware might hide its true payload to deceive the analysis results, terminate its execution or remain completely dormant and even destroy any possible execution evidence [32]. Researchers have approached this issue with an effort and emphasis on developing more complete and comprehensive dynamic techniques with the ability to uncover and mitigate anti-forensic mechanisms [19, 21], creating stealth platforms and transparent analysis frameworks [81, 82] or detecting evasive malware behaviors and split personalities [25, 39].

Efforts have also been made towards improving the efficiency of dynamic analysis procedures [83]. However, as analysis techniques and tools evolve and become more advanced and effective, attackers also manifest new mechanisms to avoid detection and impede modern analyzers, leading to a continuous "*arms race*" between analysts and malware creators [23].

2.8.3 Reverse Engineering

Reverse engineering refers to the process of generating and analyzing the corresponding assembly instructions of a given malware [22]. This is achieved by importing the executable file into a disassembler, to produce the assembly code, and examining its instructions to discover the code's functionality.

However, the compilation of a program's source code results into a machine optimized code which is typically stored in a binary form so that it can be efficiently executed by the computer [15]. Disassembling the complied code could provide complicated and confusing assembly instructions accompanied by machine produced variable names. Moreover, as assembly instructions provide a low level representation of the original code and they are inherently dependent on the microprocessor's family, the reverse engineering process requires high expertise, advanced programing skills and extensive knowledge of system's architecture [22].

Regardless of the aforementioned difficulties, reverse engineering can provide important information about the malware's inner programming structure and logic that simple static analysis techniques fail to reveal. Figure 2.6.1 illustrates a typical reverse engineering technique that uses static extraction to produce a resulting Control Flow Graph (CFG) representation of the binary, which assists the discovery of different execution paths inside the code [84].

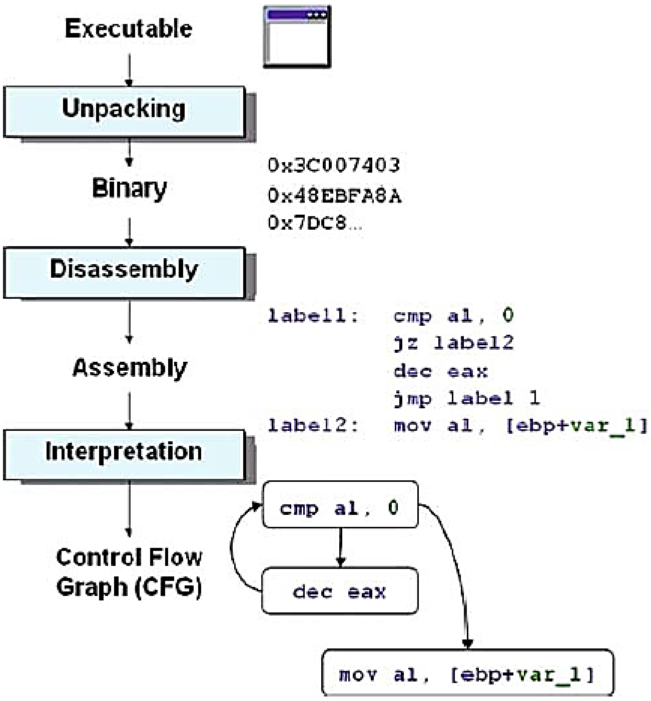


Figure 2.6.1: Static extraction reverse engineering methodology.

To hinder reverse engineering methodologies, malware authors have also developed various anti-disassembly techniques to produce false assembly listings and to frustrate any reverse engineering efforts [15, 32]. Modern analysis methodologies and tools are commonly oriented towards extenuating and overcoming such mechanisms. Recently researchers have attempted to reverse engineer the Zeus toolkit [62] using the “PaiMei” reverse engineering framework [85]. However, reverse engineering software phases some legal considerations and restrictions [15]. Before attempting to reverse engineer a program, one should be aware of the respective laws and regularities, applicable in each distinct analysis case.

2.9 Malware Analysis Tools

A variety of analysis tools exists to facilitate the entire process of malware investigation. Such tools include disassemblers, debuggers, memory managers, unpackers, network, registry and process monitors, strings searchers, text extractors and numerous others valuable in common malware analysis practices. These tools can be incorporated in various combinations to assist both static as well as dynamic analysis procedures.

However, prior to employing and engaging any type of analysis tool, important considerations should be taken into account regarding their validity and forensic acceptability [19]. The pool of existing tools includes various commercial, open source or freeware packages in addition to multiple others that can be retrieved from numerous on-line sites. The forensic soundness and legality of these tools should be thoroughly examined so as to legitimately support all stages of forensic investigations and prosecutions.

A plethora of available analysis tools as well as an extensive description and demonstration of their usage and functionalities can be found in [21, 22]. Table 2.7.1 presents an explicit example of a distinct set of tools that were combined to assist the process of a detailed analysis approach towards dissecting and examining a specific malware sample [18].

Table 2.7.1: Example of a combination of analysis tools to dissect a specific malware.

Technique	**Functionality**	**Tool**
Static Analysis	Automatically packs and unpacks malware.	UPX (*upx.sourceforge.net, 2008)*
	Searches for ASCII and Unicode strings in binary files.	Strings (s*ysinternals.com, 2008)*
	Reads files and displays documents in textual or hexadecimal format.	FileInsight *(McAfee, 2009)*
	Identifies packers, cryptors and compilers in PE files.	PEiD (*peid.info, 2008)*
	Allows viewing and editing of PE files.	Stud PE *(cgsoftlabs.ro, 2008)*
	Disassembler with graphing abilities.	IDA Pro, (*hex-rays.com, 2008)*
Dynamic Analysis	Debugger with GUI.	OllyDBG *(ollydbg.de, 2004)*
	Lists auto-starting locations	AutoRuns (*sysinternals.com, 2011)*
	Displays running processes, threads, DLLs, handles etc.	Process Explorer (*sysinternals.com, 2011)*
	Monitors and logs filesystem, registry, process and network activity.	Process Monitor *(sysinternals.com, 2011)* CaptureBAT *(honeynet.org, 2007)*

	Lists loaded DLLs.	ListDLLs *(sysinternals.com, 2011)*
	Lists all TCP and UDP endpoints.	TCPView *(sysinternals.com, 2011)*
	Displays the physical and virtual memory usage of a process.	VMmap *(sysinternals.com, 2011)*
	Displays the namespace of Windows' Object Manager.	Winobj *(sysinternals.com, 2011)*
	Extracts text.	BinText *(McAfee, 2000)*
	Displays registry and file changes by comparing two snapshots.	Regshot *(sourceforge.net/projects/regshot, 2007)*
	Identifies variations in the process' handle tables.	HandleDiff *(malwarecookbook.com, 2011)*
	Captures network traffic and analyzes packets and protocols.	Wireshark *(wireshark.org, 2010)*
	Contains utilities useful in malware analysis such as an MD5 hash calculator, FakeDNS etc.	Malcode Analysis Pack *(http://www.woodmann.com/collaborative/tools/index.php/ , 2001)*
Static and Dynamic Analysis	A lightweight Linux distribution that incorporates various tools and utilities for analyzing and reverse-engineering malware.	REMnux *(http://zeltser.com/remnux, 2011)*

To facilitate the analysis process as well as to reduce analysis time and manual effort, recent research efforts and advancements have introduced various automated malware analysis tools and platforms. These tools have the ability to perform malware analysis in an automatic manner, resulting to a set of generated reports that help analysts to identify and understand a malware's behavior. By utilizing automated tools, security experts can promptly respond to a potential threat and quickly built appropriate defenses. The following paragraphs provide a brief description of some of the most common automated dynamic analysis tools as they are presented and discussed by Egele et al. [23] as well as a brief overview of the Cuckoo Sandbox analyzer [54].

Anubis [86], which was based on TTAnalyse [12], analyzes suspicious binaries by executing the samples in a Windows XP guest OS that runs inside Qemu emulator [87]. To perform the analysis Anubis monitors and records all Windows API calls as well as the corresponding passing parameters and arguments relevant to the processes that were generated by the sample under investigation. The analysis results are stored in reports containing all recorded activities. Since the analysis was able to capture only one execution path, Moser et al. [13] extended Anubis so as to enable the exploration of multiple execution traces.

CWSandbox [88] can perform malware analysis in either a native dedicated physical machine or in a virtualized Windows environment. It implements API hooking to hook functions responsible for monitoring API calls. CWSandbox additionally applies rootkit technology to conceal the presence of the analysis tools from the investigated sample. The analysis outcome is a report that includes all the performed actions of the analyzed sample. Recently security experts developed the *GFI Sandbox*[32] automated malware analysis tool which was built based on CWSandbox. It is designed to analyze Windows executable files preferably on a native machine. GFI Sandbox can be used either as an online service[33] by submitting a sample for analysis or purchased as a commercial malware analysis package. One of its included features provides multiple malware analysis comparison between the same or different malware samples which is much similar to the current proposed approach of this dissertation. However, in contrast to our methodology, the analysis offered by GFI Sandbox is restricted into Windows files only. Moreover a single product license starts at $15.000 making it a luxurious choice as opposed to the recommended open source solution.

Cuckoo Sandbox [54] is an open source malware analysis tool which implements automated malware analysis in an isolated and controlled virtualized environment. The execution of the malware samples can be applied in various guest operating systems including multiple versions of Windows and linux distributions. Its features, among others, include monitoring and recording native functions and Windows API calls, network traffic, memory dumps, generated processes and dropped files. The functionality and behavior of each given malware sample is processed and documented in both human

[32] http://www.gfi.com/malware-analysis-tool#overview

[33] http://www.threattrack.com/

and machine readable reports to allow additional processing and investigation. It can be installed locally and further customized or used as an online analysis engine[34].

It is important to mention that utilizing automated and on-line analysis tools also comes with legal considerations regarding any confidential information that might be disclosed to an external party [19].

2.10 Overview of Malware Analysis Techniques and Methodologies

The selection of the appropriate techniques, implementation methodologies and analysis process solely resides in the analysts hands who needs to leverage between the available choices and either follow a predefined and already demonstrated procedure or deploy a custom made and tailored methodology, which would better assist accomplishing the desired goals and achieving the necessary results.

Static Analysis Techniques

Static analysis entails inspecting the malware code without executing it. To this end static analysis is safer as the malware will not be allowed to deliver its payload and realize any destructive actions. To avoid any accidental activation of the sample under investigation, using an analysis environment other than the one intended for the malware to run is usually a safe choice [43].

Different techniques and approaches are utilized to make the static analysis process feasible [22, 43], some of which are described below:

Virus Scanning. Before initiating the analysis process it would be a good practice to first check if the malware sample has already been identified and examined. Various online platforms like *VirusTotal*[35] allow the submission of suspicious files which are then scanned among multiple antivirus scan engines. The results can provide an important initial insight on the malware's behavior.

File Fingerprinting. Obtaining a cryptographic hash of the suspicious specimen is imperative before proceeding further into the analysis process. By computing the hash value of the investigative file (e.g. MD5, SHA256, SHA1) the analyst can identify other

[34] http://malwr.com/

[35] http://www.virustotal.com/

instances of the same malware, possibly using a different name, as well as to occasionally authenticate the sample in order to identify any file alternations and modifications that may have caused changes to the original malware sample.

Strings Search. Most programs contain strings (character sequences) embedded inside the executable which are usually symbolized in either ASCII or Unicode format. These strings may involve status, error or other messages that are usually outputted on the screen, URL connections or even the locations of created files. Tools like *Strings*[36] can be used to identify and extract strings from executables. By examining such strings, many features and functionalities of the sample can be revealed.

Packer Detection. Malware authors are widely employing obfuscation and packing mechanisms to confuse analysts and obstruct the static analysis process. Since legitimate software usually contains multiple strings, the absence or limited number of strings within an executable might indicate malicious intent. Various tools can identify whether a program has been packed. In such cases the sample needs to be unpacked before performing the analysis.

PE files Examination. The Portable Executable (PE) format is basically a data structure used in Windows operating systems for executable files. A variety of information such the compilation date, imported and exported functions, linked libraries, code and version details and many more, can be retrieved by analyzing the metadata that are stored inside the PE file's header. Tools like the *Depends*[37] and *PEview*[38] examine PE files and retrieve valuable information.

Disassembly. Advanced static analysis involves examining the code's inner logic (semantics) and requires extensive reverse engineering knowledge. This is accomplished with the use of a disassembler and by analysing the assembly instructions of the malware's code. A detailed description and instructions on how to disassemble a malware and interpret the assembly code, as well as insights on systems' architecture and design can be found in [22].

Dynamic Analysis Techniques

[36] http://bit.ly/ic4plL

[37] http://www.dependencywalker.com/

[38] http://www.magma.ca/~wjr/

Dynamic analysis deals with analyzing the malware's activities during execution. Various techniques are available and can be found in related literature [21, 22, 23, 38]. Some of the most important techniques related to automated dynamic analysis as well as dynamic code analysis are described in the following paragraphs.

Function Hooking. One of the most important aspects of malware analysis is the ability to monitor and record the functions that are invoked by the malicious code. This can be realized by intercepting the corresponding function calls [23], a process known as hook*ing*. For each different function call, a *hook function* is also evoked alongside. The hook function is able to examine and record the function's execution and passing parameters. This process can be applied to monitor API[39] calls, responsible for managing files or network connections, System calls as well as Windows Native API calls. Function hooking can be implemented in different ways:

- If the source code is available, calls to the hook functions can be attached directly inside the program.
- If the sample is in binary form the one approach would be to rewrite the investigated function so as to call the hook before its execution.
- Replacing the binary's invocation instructions so as to call the hook function instead of the intended one.
- Insert breakpoints, either inside the monitored function or at each invocation command to that function so that on each breakpoint's activation, a debugger will be executed to control the entire process.

Function Parameter Analysis is used in order to examine the values of the invoked function's passing parameters. Tracing input and return values allows clustering function calls into groups of similar functionality.

Information Flow Tracking. The purpose of this technique is to provide an understanding on how the corresponding data of a program are circulated inside the infected system throughout its execution. Following the traces of the data can provide significant information on the processing behavior of the sample. To achieve this, the monitored data are tainted with a distinct label. This label is propagated whenever the data are processed by the program and the information flow is monitored and recorded.

[39] Application Programming Interface

Instruction Tracing is applied in order to better realize how the analyzed sample behaved during execution. Instruction trace refers to monitoring and analyzing the order of the sample's executed instructions during the analysis.

Autostart extensibility points (ASEPs) are techniques used to automatically execute a program upon the system's reboot process or upon the execution of an application. Malware samples most often attach themselves to existing ASEPs so as to be executed automatically. Monitoring and analyzing such mechanisms is imperative in order to understand inherent malware behavior and infection strategies.

Implementation methodologies.

Designing an appropriate system in which to execute the malware sample and to perform the analysis techniques is a complicated and difficult task that can significantly affect the resulting outcomes. Important consideration should be given to the defined privilege levels in the execution environment. There are mainly three available approaches [23] to implementing a dynamic analysis system:

Kernel or User Level implementation entails using a native dedicated system to perform the analysis. Executing the malware in user-space allows capturing a wide range of information, such as for example all types of invoked calls and executed processes, as the same operating system's recourses are available to all implicated applications. The main limitation of this type of implementation is that the analysis modules can be easily detected by the malware. Executing the analysis modules with kernel level privileges provides additional information, like on specific system calls, and can conceal the usage of the analysis tools. However, malware that can elevate privilege levels will still be able to detect the analysis mechanisms.

Implementation in an Emulator allows the execution of the malware sample inside a controlled simulated environment. Specialized software, like for example Qemu [87], can emulate parts or the entire structure of a personal computer including the operating system and applications as well as the system's hardware recourses such as the processor, storage disks, peripherals and more [12]. The operating system inside the emulator is referred to as the guest OS. Additionally, the host and the guest architectures could be different. This type of implementation, depending on the level of emulation, can provide full control over the test environment and prevent malware from detecting it. However, advanced malware instances have the ability to identify specific emulation characteristics and terminate its activity. Moreover, significant high level information such as system or function calls need to be deduced from raw system data.

Implementation in a Virtual Machine. With this approach the malware is executed inside an isolated virtualized environment. Virtualization software, like VirtualBox [89], can simulate various operating systems and assign subsets of the host system's hardware recourses to the running guests [12]. The management and control of the virtual machines is performed through a Virtual Machine Monitor (VMM) which is responsible for allocating system resources to the guests [23]. After each malware execution, the related VM can be reinstated to an uninfected status using clean VM snapshots. In general, virtualization enables fast malware analysis mechanisms and provides full control over the analysis environment. As opposed to emulation, it uses the host's physical hardware resources to execute the malware's instructions. However, the resources are strongly isolated, thus concealing the analysis tools from the malware. Nevertheless, malicious authors have also manifested techniques to detect virtualized analysis environments [32, 39].

Analysis Methodologies

Early manual analysis methodologies like the one proposed by Skoudis and Zeltser [15] were quite straightforward and solid, involving a successive number of steps while combining both static and dynamic analysis techniques. The authors illustrate a detailed analysis concept, were the malicious sample can be examined either in a physical dedicated test laboratory or in a virtualized environment, and additionally provide a malware analysis template, which can be used for the purpose of recording observed results during the analysis process. The respective template including all necessary activities that need to take place has been revived in the following table:

Table 2.1: Malware Analysis Template provided by Skoudis and Zeltser.

Activity	**Observed Results**
Load specimen onto victim machine	
Run Antivirus program	
Research antivirus results and file names	
Conduct Strings analysis	
Look for scripts	
Conduct binary analysis	
Disassemble code	
Reverse-compile code	
Monitor file changes	

Monitor file integrity
Monitor process activity
Monitor local network activity
Scan for open ports remotely
Scan for vulnerabilities remotely
Sniff network activity
Check promiscuous mode locally
Check promiscuous mode remotely
Monitor registry activity
Run code with debugger

The above methodology presents a rather linear attitude and does not focus on providing the means towards overcoming possible anti analysis techniques that the malware might engage.

In 2007 Zeltser introduces a new methodology [17] which enhances the possibility of revealing and overcoming modern anti forensic techniques. Zeltser proposes a series of repetitive steps, with the purpose of extracting the complete functionality of the underlying code, which are summarized and respectively reproduced as follows[40]:

- *Step 1*. Set up a controlled, isolated environment to perform the analysis.
- *Step 2*. Examine the malware's behavior through behavioral analysis.
- *Step 3*. Conduct static code analysis to understand the code's inner-structure.
- *Step 4*. Perform dynamic code analysis to get additional information on the code.
- *Step 5*. If packed, unpack the sample
- *Step 6*. Execute steps 2, 3, and 4 repetitively until the analysis goals are met.
- *Step 7*. Record results and clean the test environment for future analysis.

The proposed methodology blends together static and dynamic analysis techniques and, as opposed to the linear nature of Skoudis' approach, through an iterative and recursive procedure dives deeper into the analysis process from a higher and abstracted level view to a more detailed and refined view. After each analysis cycle the test platform is molded and tailored based upon specific findings and behavioral observations, in order to promote interactions with the specimen, stimulate additional malware activity and unfold possible hidden aspects of its malicious intentions. Moreover, through the dissemination of behavioral and code analysis techniques that interchange and intertwine

[40] http://zeltser.com/reverse-malware/reverse-malware-cheat-sheet.html

throughout the analysis process, Zeltser's method also facilitates the opportunity to discover and mitigate anti forensic techniques as the analysis process proceeds [19].

Distler in "Malware Analysis: An Introduction" [14], building on Zeltser's technique, presents an analysis methodology were both static and dynamic techniques are incorporated. The author implements a malware lab for the analysis which consists of four iconic systems created with VMware virtualization software and exhibits a step by step examination procedure by employing several manual analysis tools in order to assist every stage of the analysis process. Distler thoroughly and practically demonstrates how one can create a sandbox environment, execute the malicious code, monitor malware's activity, collect the appropriate data and analyze it accordingly.

The proposed method comprises of a series of actions that can be broken down and distinguished depending on their involvement in the preparation, static or dynamic analysis phases. The preparation phase consists of malware acquisition, virtual lab preparation, copying, extracting and installing, when required, the appropriate tools into the virtual machines, taking MD5 hashes of the tools, base lining the system and taking a snapshot of the VM. It is suggested that upon proper preparation, multiple AV software should be executed to determine whether the suspicious file can be detected as malicious. During static analysis the malware should be first examined in a hex editor to define its type and determine the possible usage of a packer utility (like UPX). If a compression mechanism is identified, a copy of the original file must be kept before decompressing or unpacking the program. For the remaining code analysis process, following steps involve performing a strings search, disassembling and reverse engineering the malware. For the dynamic part of the analysis the suggested steps involve:

- Update and install all necessary applications, service packs, patches and hot fixes.
- Set VM networking to "Host-Only" networking.
- Perform one more Baseline, upon finishing the code analysis.
- Run Process Explorer, TCPview, Windump and explorer
- Execute the malware
- Monitor and record system status changes
- Take another snapshot using WinAnalysis
- Compare Snapshots

- Run PE and TCPview one more time and identify changes from the previous baseline
- Examine Network Trafic
- Identify and observe new processes installed by the malware
- Search for listeners
- Adjust Environment
- Inspect network traffic again

Distler's approach sketches, in a clear and solid manner, a complete analysis methodology which includes all the necessary courses of action from the preparation of the sandbox environment to the analysis of the acquired results. The proposed methodology is much similar to the one discussed also by Hutcheson in "Malware Analysis the Basics" [16]. Hutcheson additionally implements an initial visual analysis, prior to conducting static and dynamic analysis, aiming to gather some primary data involving the malicious file. In both techniques, analogous to Zeltser's method, in order to trigger additional activity and unveil as much of the malwares' personality as possible, through the process of multiple executions, the test environment is adapted and system parameters are furthermore customized depending on behavioral findings after each code run. The difference can be found in the process where part of the functionality of the investigated malware is revealed by comparing snapshots of the system before and after execution, a function which is not implemented by Hutcheson.

The aforementioned analysis approaches [14, 16, 17] provide an initial attempt towards alleviating some of the anti-detection mechanisms employed by malicious authors, such as code obfuscation and encryption.

Aquilina et al. in 2008 recommend file profiling, as an essential preliminary phase during the initial analysis procedure [21], in which static analysis constitutes a basic process module, and entails the inspection of the suspicious file in an effort to acquire significant information surrounding the malware thus leading to more accurate and targeted decisions on selecting the most suitable analysis approach. In general the profiling stages include:

- *Detail*. Detect and record the system details from which the suspect file was obtained.
- *Hash*. Acquire the cryptographic hash of the suspicious file.
- *Compare*. Perform a similarity assessment against known samples.

- *Classify*. Identify the target architecture, format of the file, authoring language and the compiler used.
- *Scan*. Use anti-virus and anti-spyware tools to determine the existence of a pre-identified malicious signature.
- *Examine*. Determine whether the sample has malicious intentions through appropriate analysis tools.
- *Extract and Analyze*. Mine strings, and discover file metadata and symbolic information.
- *Reveal*. Reveal armoring and code obfuscation techniques.
- *Correlate*. Define the existence of static or dynamic linkage of the file.
- *Research*. Perform online research to find out if the file has already been analyzed.

For the subsequent dynamic analysis phase, the authors propose the execution of the following successive actions:

- Establishing the Environment Baseline
- Pre-execution Preparation
- Executing the Malicious Code Specimen
- System and Network Monitoring
- Environment Emulation and Adjustment
- Process Spying
- Defeating Obfuscation
- Decompiling
- Advanced PE Analysis
- Interacting with and Manipulating the Malware Specimen
- Exploring and Verifying Specimen Functionality and Purpose
- Event Reconstruction and Artifact Review

Through a number of case scenarios, the proposed methodology is practically applied in both Windows and Linux based environments and depending on the investigative surroundings a plethora of manual analysis tools is utilized and a hands-on demonstration of their usage takes place.

The above mentioned methodology shares many similar characteristics with Zeltser's approach [17], like the adaptation of the test system based on intermediary analysis observations and interactions with the malware, but also additionally deploys explicit steps and formulas focusing specifically on defeating possible shielding and defensive mechanisms employed by the malicious specimen.

Brand, Valli and Woodward in 2010 extend Zeltser's methodology [17] and introduce a spiral analysis model [20], which alternately uses static and dynamic techniques, with additional emphasis on unmasking and extenuating anti analysis tactics. This enhanced method embeds a number of consequent phases which are imprinted in the following lines:

- Preliminary Static Analysis
- Tailor Static Analysis Environment
- Detect and Mitigate Static Analysis Avoidance Technique
- Detailed Static Analysis
- Preliminary Dynamic Analysis
- Tailor Dynamic Analysis Environment
- Detect and Mitigate Dynamic Analysis Avoidance Technique
- Detailed Dynamic Analysis

These subsequent steps are repeated in an iterative and recursive manner as the examination process continues and based on the obtained results after each phase appropriate decisions are made on how to tailor and adjust the following stage. Frankie Li in his technical paper "A Detailed Analysis of an Advanced Persistent Threat Malware" [18] adopts and applies Zeltser's technique in an effort to analyze and dissect a specific malware sample by conducting a detailed analysis in a spiral way [20].

A complete, hands-on practical guide to manual analysis can be found in "Practical Malware Analysis" by Sikorski and Honig [22]. The authors implement and exhibit an analysis methodology which embraces some of the most modern techniques and tools concerning static and dynamic analysis methods. The recommended approach starts with a primary basic static and dynamic analysis of the malicious file and moves on to more advanced and sophisticated static and dynamic procedures. The initial static analysis consists of using antivirus tools, hashes and gathering information from strings, functions and headers while the dynamic part implicates running the code in a virtualized environment and monitoring processes, registry and file system changes, simulating a network and sniffing packets. Following this initial investigation, advanced reverse engineering techniques are used in order to disassemble the binary and extensive debugging mechanisms and tools are employed in order to obtain a vibrant picture of the program as it is executed.

What has become evident through the aforementioned approaches is that modern manual analysis methodologies append an extra layer into the analysis process and understanding, detecting and escaping anti forensic mechanisms has become a prerequisite towards developing and implementing a fully functional and effective forensic analysis methodology.

Modern analysis methodologies lean towards implementing automated analysis techniques or integrating already existing automated analysis tools in order to form more complete and comprehensive automated frameworks and reduce analysis time and manual effort.

Ligh et.al [90], present an automated analysis methodology and introduce various python modules that can be utilized to automate several aspects of the analysis process. The proposed scripts and tools facilitate executing and monitoring malware inside virtualized environments as well as in physical machines. Figure 2.8.1 illustrates the suggested methodology, as it is presented by the authors, and demonstrates how to perform malware analysis in a reusable automated sandbox environment.

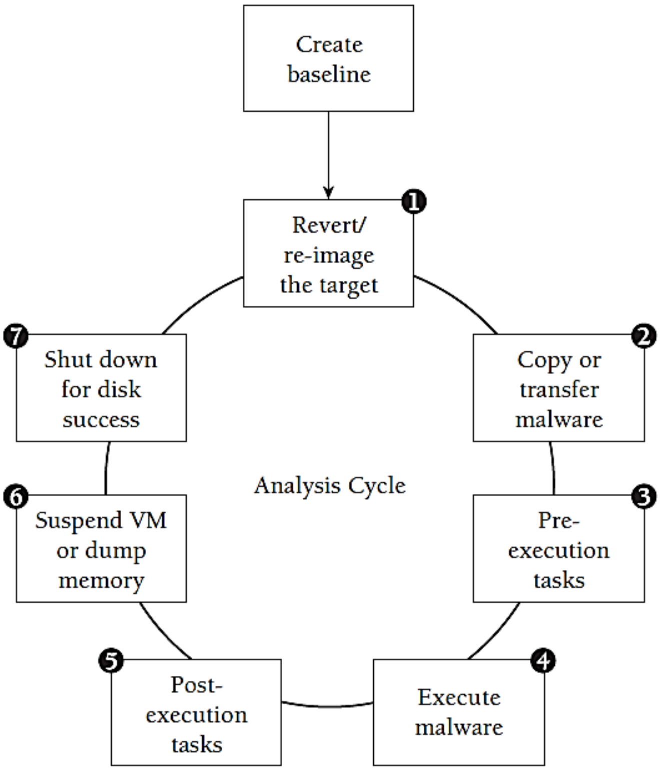


Figure 2.8.1: Automated malware analysis methodology

Each one of the distinct analysis steps can be automated through appropriate python scripts. For an automated analysis without any programming requirements, the authors propose ZeroWine[41] and Buster Sandbox Analyzer[42] as a pre-constructed solution.

3 Chapter 3 - Setting up the Test Bed

The design of the proposed malware analysis framework relies on the proper configuration and deployment of a safe and reliable Test Bed that will allow the execution of various malware samples multiple times. Cuckoo [54] is incorporated within the framework, as the main automated malware analysis tool, to produce the necessary primary data for analysis. The following sections provide a brief description of Cuckoo's functionality as well as the specifications of the hardware and software requirements that are imperative in order to create an isolated and controlled environment for testing multiple malware.

3.1 Platform Requirements

The Test Bed operates as the main analysis platform in which various malware samples can be executed and examined multiple times. The corresponding analysis results are utilized as primary data which are further correlated and processed to produce comprehensive malware profiles. To ensure the correctness of the analysis process and the accuracy of the analysis results, certain conditions and requirements need to be taken into consideration during the design of the Test Bed and the selection of the appropriate underlying infrastructure.

[41] http://zerowine.sourceforge.net/

[42] http://bsa.isoftware.nl/

3.1.1 Hardware Requirements

The required hardware infrastructure, for the deployment of a reliable and efficient malware analysis platform, mainly depends on the system's purpose and functionality and could range from a plain personal computer with minimal characteristics to more powerful and multiplicate machines. An appropriate infrastructure is imperative in order to enhance the system's performance and support the successful accomplishment of the desired goals.

For the implementation of the proposed malware forensics framework, a simple dedicated physical machine has been accordingly set-up and configured. The recruited machine is a personal computer with a Pentium IV at 3.8 GHz processor and an Asus P5 GV-MX motherboard. A hard drive of 160 GB storage capacity and a 3.5 GB RAM have been attached to support all required tools, applications and analysis activities. Additionally, an external 250 GB hard drive has been used for holding regular backups and disk images. The aforementioned hardware choices have been found to adequately support all necessary applications, storage and processing requirements of the introduced analysis framework.

3.1.2 Software Requirements

The proposed malware forensics framework utilizes Cuckoo Sandbox Analyzer [54] as the main malware analysis tool. Therefore the selection of appropriate software, complementary tools and applications was primarily based on Cuckoo's software dependencies as well as required components, libraries and modules. Any additional packages were chosen on the basis of Cuckoo's support and proper assimilation.

The Test Bed uses Ubuntu 12.04 LTS Linux distribution[43] as the underlying host operating system, with a running installation of Cuckoo version 0.4, and VirtualBox [89] as the main Virtual Machine emulator. Since Cuckoo's host modules are written in python, the corresponding python version 2.7[44] has been installed on the host. To stimulate additional features and to explore Cuckoo's full analysis potentials, several other python libraries have been utilized such as:

- *Magic*: To identify various file formats.

[43] http://www.ubuntu.com/

[44] http://www.python.org/

- *Pyssdeep*: To compute the ssdeep fuzzy hash of files.
- *Dpkt*: To retrieve network traffic information from PCAP files.
- *Mako*: To form the HTML reports and Cuckoo’s web interface.
- *Pymongo*: To store the analysis results in a MongoDB database.
- *Image Library*: To capture guest desktop screenshots during the analysis.

Tcpdump has been installed in order to capture and record any network activities performed by the malware during execution. Tcpdump works as a network sniffer and is responsible for monitoring network traffic and dumping it on a .pcap file for further processing.

In order for VirtualBox to function properly, the corresponding Software Developer Kit (*SDK*) extension package has also been installed on the host whilst the required VirtualBox guest additions have been set up inside each guest operating system.

To correctly set up and prepare the Test Bed for conducing malware analyses, all the aforementioned tools and applications along with possible available patches, service packs, updates etc. need to be installed and suitably configured. For the completeness of the proposed framework some additional libraries have been included. *Numpy, Scipy*[45] and *Matplotlib*[46] serve as scientific computing tools that facilitate the automatic charts generation feature of the introduced analysis methodology.

3.1.3 Virtual Machines

VirtualBox provides complete and flexible software-based or hardware assisted virtualization solutions, where multiple guest operating systems can be concurrently installed and manipulated on a single host machine. It additionally enables safe communication between the host and the isolated virtual guests, allowing full control and independent management abilities. The guest operating systems running inside VirtualBox, operate as potential victim machines in which various malware samples are executed and analyzed.

To demonstrate the functionality and effectiveness of the proposed analysis approach, four differently configured virtualized environments have been instrumented based on

[45] http://numpy.scipy.org/

[46] http://matplotlib.org/

two different operating systems. These include a clean installation of Windows XP SP3 along with a second duplicate operating system enriched with various applications as well as two differently configured and software populated Windows 7 systems. After the appropriate preparation and customization of the guest systems, each virtual machine is cloned and separate "clean" snapshots are taken through either the Virtual Machine Monitor or the VBoxManage utility provided by VirtualBox. These snapshots are later on utilized so as to reinstate the machines to their previously uninfected status, after each analysis run.

The selection of the appropriate test systems and their internal configuration is neither exclusive nor restrictive and can be adjusted to suitably reflect each specific organizational context for which the malware investigation is being conducted.

3.2 Working with Cuckoo

Cuckoo Sandbox [54] is a newly developed open source software package which incorporates fully automated malware analysis features, providing fast and complete analysis results. Cuckoo's components are written in python making them fully customizable and extensible so as to serve specific analysis goals and requirements. Moreover, Cuckoo can be employed either as a standalone analysis tool or integrated within broader investigative procedures facilitating the development of more coherent and comprehensive analysis frameworks.

Cuckoo is designed to automatically execute and analyze suspicious files inside isolated environments. Each malware sample is separately executed on the guest machines and the entire analysis process is managed through the core components that run on the host. An agent running inside the guests undertakes the communication with the host. Furthermore, Cuckoo embodies stealth characteristics, as it employs rootkit-based technology to safely perform the analysis procedures and conceal its activities from the malicious samples under analysis. Figure 3.1 presents Cuckoo's basic architecture.

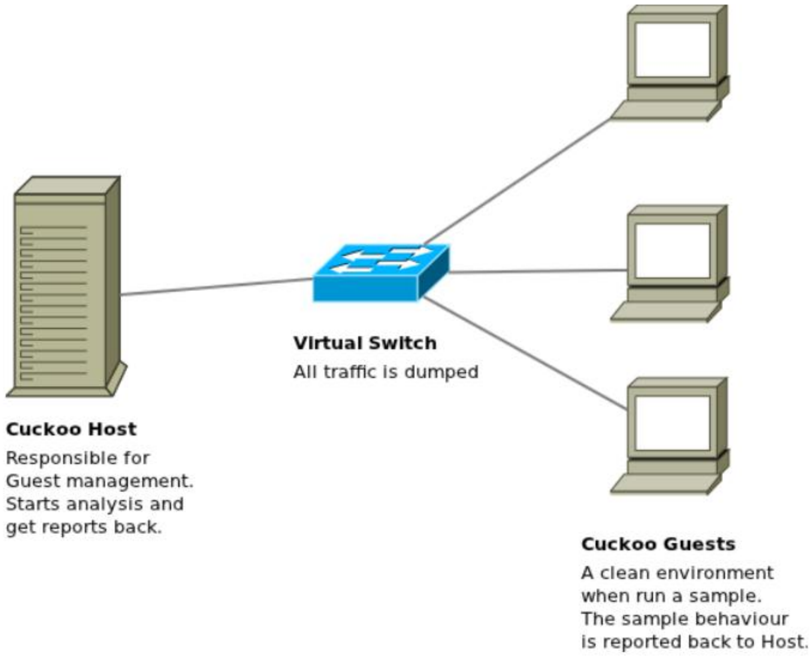


Figure 3.1: Cuckoo's Architecture

Before submitting any samples for analysis, Cuckoo's configuration files need to be modified to comply with the underlying system specifications, Virtual Machines' characteristics and individual user settings.

Cuckoo analyzer can be activated by executing the command *python cuckoo.py* through a terminal window. The analyzer is launched inside the host and retrieves samples for analysis from the available Cuckoo SQL database. The submission of malware samples can be done either through Cuckoo's provided web interface or from a new terminal window through the *submit.py* utility. Samples can be submitted in random order multiple times with various analysis specifications. Even though this process is currently performed manually, it can be automated using appropriate python scripts. All analysis requests are stored inside Cuckoo's DB with a distinct analysis ID.

The analyzer retrieves the analysis requests from the database and performs the analysis independently for each malware sample. Cuckoo has the ability to execute and analyze multiple suspicious files concurrently. For each analysis process, separate subfolders are created to hold all available analysis information and observed malware behavior. Each subfolder is named after the distinct ID of the specified analysis request. After the completion of each analysis procedure, the results are stored in the respective subfolder. These results include:

- An analysis.log file with information relevant to the analysis process.
- A dump.pcap file with the recorded network traffic.
- All files that were manipulated by the malware.
- Row CSV log files that contain all generated processes and relevant API calls.
- Various reports documenting the malware's behavior in both human and machine readable formats.
- Various screenshots captured during the malware's execution inside the guest.

During processing Cuckoo holds the observed malware activities in a "*Global Container*" with a *json-like* format, which is basically a large python dictionary. This dictionary is used to produce all related html, json, pickle, and xml formatted reports. Cuckoo additionally adopts MAEC[47], the new standardized language for malware characterization, and generates the respective maec11.xml report, thus enabling the correct and accurate communication of malware behavioral attributes and artifacts. Optionally the information within the global container can be stored in a MongoDB[48] database to enable further querying and processing capabilities.

[47] https://maec.mitre.org/

[48] http://www.mongodb.org/

4 Chapter 4 - Experimentation

After the successful preparation of the Test Bed various malware types will be acquired and tested against different systems. This chapter will contain a description of our research and experimentation process.

After each malware testing procedure, the platform as well as the quest operating systems will be returned to their original “clean” state in order to ensure the accuracy of our test results for each virus. The research and experimentation process is expected to be conducted until the beginning of August.

4.1 Malware Acquisition

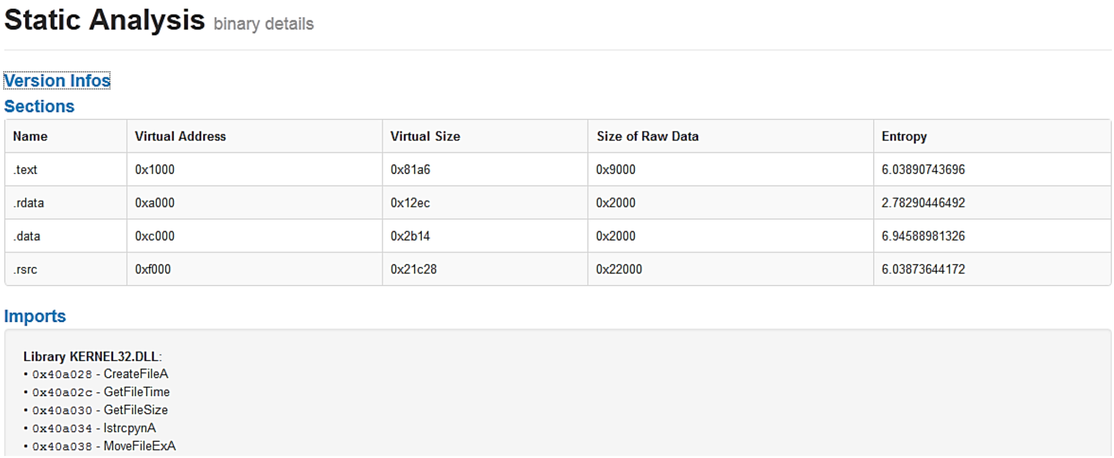
Offensivecomputing.net will be used in order to acquire and test multiple viruses. This section contains and describes the procedure of malware acquisition and preparation.

4.2 Testing Malware Behavior in Different Operating Systems

Using Cuckoo as our main malware analysis tool, each acquired virus will be tested in different environments and the analysis results will be recorded in an appropriate format for further study.

4.2.1 Static Analysis

For each malware the static analysis procedure will be described and the corresponding results will be documented.



Static Analysis binary details

Version Infos

Sections

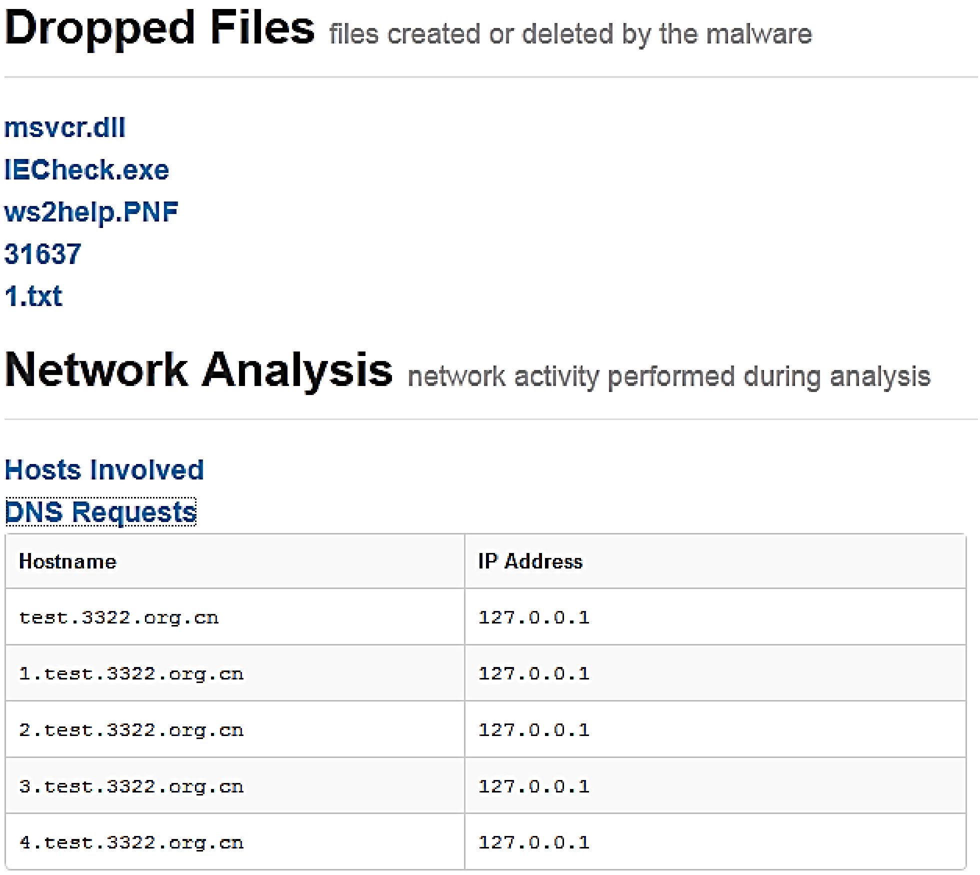
Name	Virtual Address	Virtual Size	Size of Raw Data	Entropy
.text	0x1000	0x81a6	0x9000	6.03890743696
.rdata	0xa000	0x12ec	0x2000	2.78290446492
.data	0xc000	0x2b14	0x2000	6.94588981326
.rsrc	0xf000	0x21c28	0x22000	6.03873644172

Imports

Library KERNEL32.DLL:
- 0x40a028 - CreateFileA
- 0x40a02c - GetFileTime
- 0x40a030 - GetFileSize
- 0x40a034 - lstrcpynA
- 0x40a038 - MoveFileExA

4.2.2 Dynamic Analysis

The dynamic analysis procedure and results for each malware under investigation will be described in this section.



Dropped Files files created or deleted by the malware

msvcr.dll
IECheck.exe
ws2help.PNF
31637
1.txt

Network Analysis network activity performed during analysis

Hosts Involved
DNS Requests

Hostname	IP Address
test.3322.org.cn	127.0.0.1
1.test.3322.org.cn	127.0.0.1
2.test.3322.org.cn	127.0.0.1
3.test.3322.org.cn	127.0.0.1
4.test.3322.org.cn	127.0.0.1

5 Chapter 5 - Experimental Results Analysis

The entire experimentation phase will be thoroughly monitored in order to gain a deep insight and understanding of each malware behavior. During and after the completion of each test phase, the recorded and observed results will be analyzed and compared. This research phase will be conducted in parallel to the experimentation process.

This chapter includes the description of the manual and automatic analysis procedures of the experimental results of each malware execution, as well as the development of a research methodology that will allow us to compare malware conduct and identify possible behavioral differences in malware performance among different systems.

5.1 Manual Observation of Malware Behavior

Malware conduct as well as static and dynamic analysis results produced by Cuckoo will be manually and empirically observed in order to identify malware specific performance, file system and registry changes in each system, network traffic will be monitored etc.

This process will help us to manually identify malware behavioral differences in various systems.

5.2 Automated Analysis of Malware Behavior

An effort will be given in order to develop a methodology for a more automated malware analysis procedure and comparison of the results among the malware specific behavior in different environments. A methodology will be developed that will automatically compare the analysis results that are produced by Cuckoo and identify possible differences in its behavior.

This new research approach and methodology will be described and discussed in this section.

After the execution of dg003.exe malware sample, profiler automatically generated a txt report containing all analysis information in a human readable format.

The following txt segment displays some initial abstracted information regarding the results from the multiple executions of the sample.

```
================================================================
 Analysis of dg003.exe
 MD5 4ec0027bef4d7e1786a04d021fa8a67f
 Total Executions:    11
================================================================

================================================================
 Content Menu
================================================================

    1. General information
    2. Dropped files
        15 Total different dropped files
        3 Files were dropped in all executions
        9 Files were dropped only in one execution
        3 Files were dropped in various executions
     Files dropped in all executions
        2.1 File: ws2help.PNF
        2.2 File: 11025
        2.3 File: 1.txt
     Files dropped in one execution
        2.4 File: msvcr.dll
        2.5 File: msvcr.dll
        2.6 File: msvcr.dll
        2.7 File: msvcr.dll
        2.8 File: msvcr.dll
        2.9 File: msvcr.dll
        2.10 File: msvcr.dll
        2.11 File: msvcr.dll
        2.12 File: msvcr.dll
     Files dropped in various executions
        2.13 File: IECheck.exe
        2.14 File: netstat.exe
        2.15 File: msvcr.dll
    3. Network analysis
        3.1 DNS requests
        3.2 HTTP requests
    4. Behavior analysis
        4.1 Process: dg003.exe (1768) Found in all executions
                 2178 Total different API Calls
                 311 APIs found in all executions
                 31 APIs found in only in one execution
                 1836 APIs found in various executions

================================================================
```

5.3 Malware Behavior Comparison

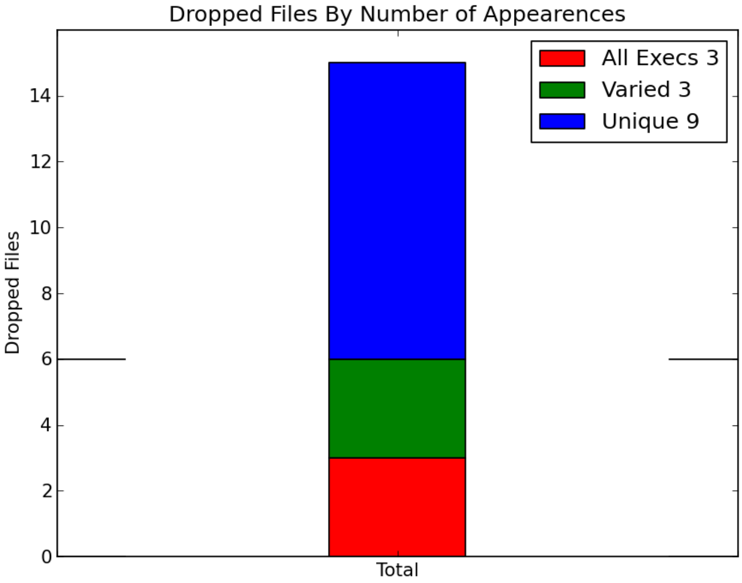
Based on both manual and automatic analysis procedures, the results will be compared and the outcome will be discussed and documented.

```
[2.2] "11025":
  File size: 36864 bytes
  File type: PE32 executable (console) Intel 80386, for MS Win-
dows
  CRC32:     81D040BE
  MD5:       8a7ee413726790398d6b315b7cfb5b0a
  SHA-1:     43a10634c617ce6f4d598a6f3ca3d5fe403d986c
  SHA-256:
958eb25df9d1f1f1cf807b9a6efe6041d93885ccedc3f6a2f3cbb113ffc842ac
  SHA-512:
674c9c1148dffee67c37a2e2693bbebd2d95c43ce1c88115086717bfad9f6d09
fb7679c48e6f416a95799e2f061ce9735da7ad5d644858f38e87b780f3a414b7
  Ssdeep:    None
  Same as :
             1  "18114" in analysis : "5028cbb6f489330c7600006e"
             2  "19208" in analysis : "5028cc76f489330c76000079"
             3  "28198" in analysis : "5086b353f489330a70000066"
             4  "1430" in analysis : "5087cce6f489330d7100006a"
             5  "10899" in analysis : "508a99cff48933160d00006e"
             6  "12447" in analysis : "508a9ba8f489331650000072"
             7  "13897" in analysis : "508a9cd3f4893316a900000c"
             8  "9485" in analysis : "508b0db6f48933240100006c"
             9  "26150" in analysis : "508b6ffef4893309ce00005c"
             10  "31637" in analysis: "508c3a86f489331814000064"
```

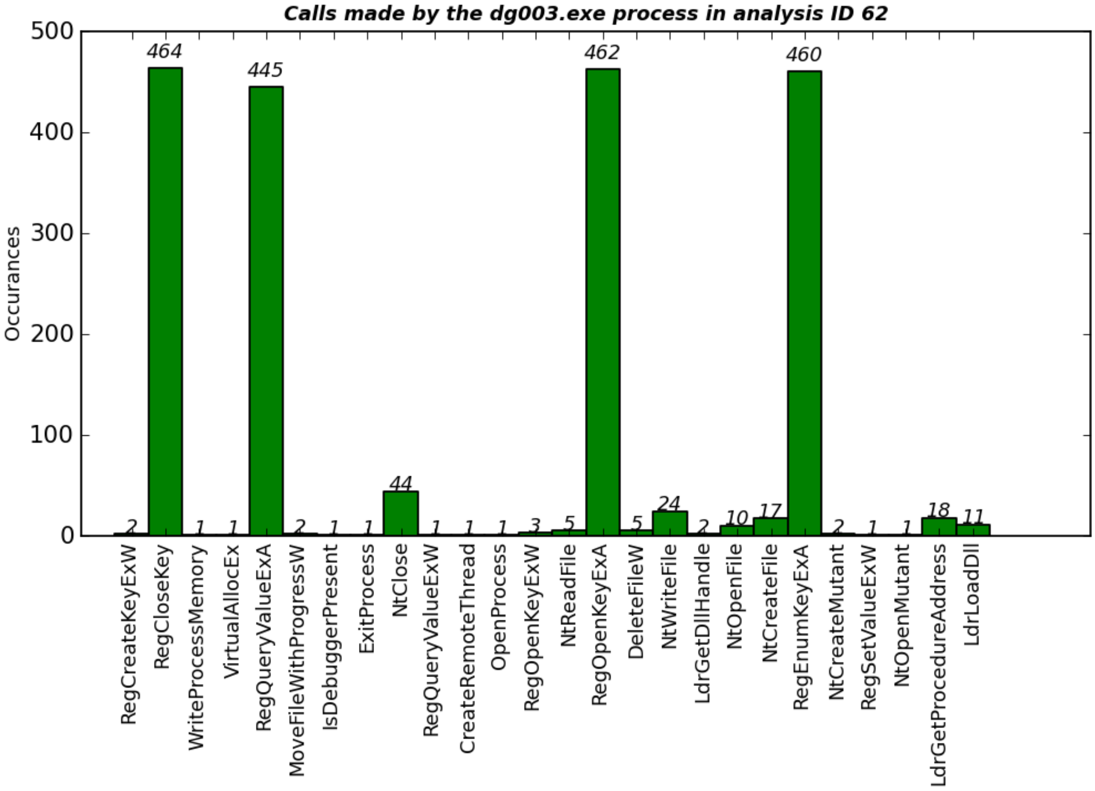
5.4 Identifying Behavioral Differences

Using all aforementioned techniques we will be able to draw conclusions and identify possible malware behavioral differences deriving from both manual observation and automated analysis procedures.

These differences in the malware conduct between different systems will be recorded, analyzed and discussed in this section.

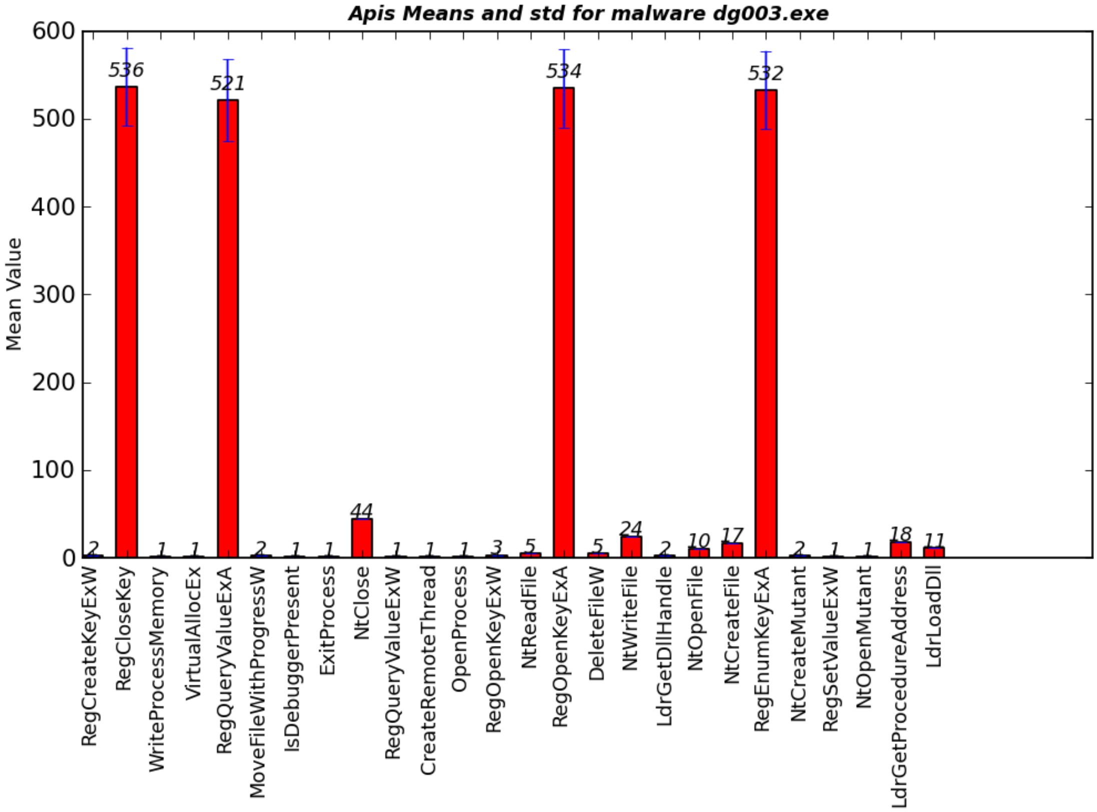


Dropped Files By Number of Appearences



Calls made by the dg003.exe process in analysis ID 62

**Total Calls 1985*


Apis Means and std for malware dg003.exe
Mean Value
0
100
200
300
400
500
600
2
536
1
1
521
2
1
1
44
1
1
1
3
5
534
5
24
2
10
17
532
2
1
1
18
11
RegCreateKeyExW
RegCloseKey
WriteProcessMemory
VirtualAllocEx
RegQueryValueExA
MoveFileWithProgressW
IsDebuggerPresent
ExitProcess
NtClose
RegQueryValueExW
CreateRemoteThread
OpenProcess
RegOpenKeyExW
NtReadFile
RegOpenKeyExA
DeleteFileW
NtWriteFile
LdrGetDllHandle
NtOpenFile
NtCreateFile
RegEnumKeyExA
NtCreateMutant
RegSetValueExW
NtOpenMutant
LdrGetProcedureAddress
LdrLoadDll

6 Chapter 6 - Malware Forensics Framework

This chapter provides a detailed description of the design, architecture and implementation aspects of the introduced malware forensics framework. The functionality of the framework as well as its internal characteristics and processing components are presented and discussed. The proposed malware forensics framework was mainly conceived and developed with the purpose of providing a holistic and comprehensive view on the behavioral aspects of malicious creations so as to allow forensics analysts to offer an opinion as to how a specific malware behaves under a certain environment as well as the ability to identify malware behavioral differences and similarities among variant systems.

6.1 A Malware Forensics Framework Proposal

Modern malicious instances are characterized by composite behavior and functionality. Certain behavioral aspects might be triggered only upon the acknowledgment of specific environmental parameters while performance variances could differently affect each infected machine. Therefore a single execution of a given malware can only reveal the behavior of the sample under those specific system conditions. To stimulate and monitor multiplicate behavioral characteristics, as well as to understand how the sample acted within a desired organizational context, the proposed analysis process entails executing the same malware sample multiple times in variant simulated platforms.

The basic idea behind our malware forensics framework proposal resides on finding an appropriate method for collecting, processing and correlating the analysis results as they are produced by Cuckoo Sandbox analyzer.

As mentioned earlier, Cuckoo processes malware samples and stores the results in an analysis results folder. For each analysis request, Cuckoo creates a separate subfolder containing all generated human and machine readable reports, raw log files, .pcap files, screenshots, and any other information captured during the analysis. Samples may be submitted multiple times, randomly, and subfolders are named after each analysis distinct ID. Optionally Cuckoo stores the results in an analysis collection inside a MongoDB database for further querying and processing activities. The results are stored based on the submission order of the analysis requests.

The first step towards implementing our framework is to decide how to collect the primary data for analysis. Three different options seem to be available:

- Retrieve the machine readable reports from the analysis storage folders.
- Collect and process the raw analysis results from the respective folders.
- Connect to the MongoDB and retrieve stored data.

Collecting and analyzing the raw CSV logs and .pcap files would initially require their appropriate transformation to a processible data structure format. Even though this approach could enhance the framework's flexibility and eliminate possible Cuckoo dependencies, the fact that Cuckoo already processes and converts the results to a machine accessible format in conjunction with specific restrictions with regards to the duration of the framework's development and implementation period, led to the selection of a different data collection methodology.

The json-like generated reports enclose the analysis results obtained from Cuckoo's global container, formatted as a dictionary data structure. However the location of the results folder depends on the original Cuckoo installation path which can be user defined. On the other hand, the MongoDB's installation path is standard and resides inside a predefined system's folder. Since the analysis results stored in the MongoDB database follow the exact same dictionary format, utilizing the storage, querying and processing capabilities provided by the MongoDB would significantly enhance the overall effectiveness and usability of the proposed framework. Therefore, the collection of the primary data for analysis for the current implementation of the proposed framework is performed through the MongoDB database collections.

Figure 6.1 presents the general architecture of the suggested malware forensics framework. The malware samples can be submitted for analysis multiple times and in random order. Cuckoo retrieves the analysis requests from the SQL database, executes the samples inside the guest VMs, stores the results in the respective folders as well as in the mondodb and communicates the analysis status back to the SQL database.

Profiler, the framework's core processing module, runs independently of any Cuckoo related actions. It's design and implementation is completely python based, thus allowing it to be fully customizable, modular and platform independent. It can be installed to any desired location inside the host machine and executed at any point. The installation process does not require any additional configuration on the host system and can be achieved simply by copying the profiler's python script and its additional components inside any folder and at any location in the host. Since Cuckoo's SQL database can be located anywhere inside the host, the only current requirement is to place a copy of the SQL database inside the running directory of profiler. If the location of Cuckoo inside the host is already known and predefined, profiler can be customized so as to directly connect to the SQL database.

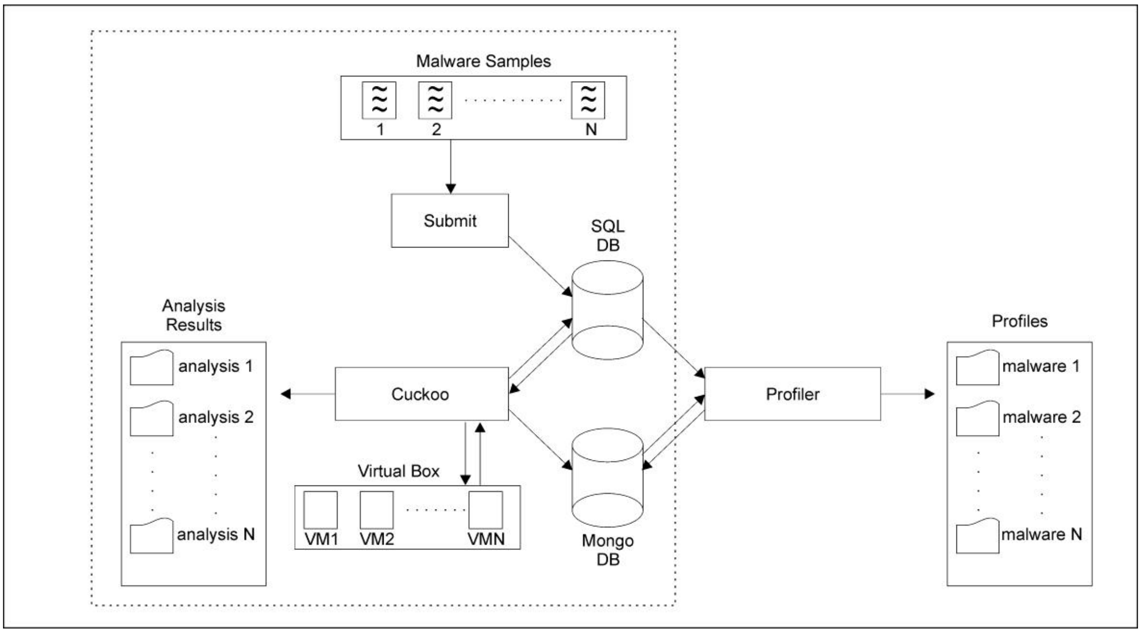


Figure 6.1: Malware Forensics Framework Architecture.

Profiler can be executed through a terminal window by running the command *$ python profiler.py* inside profiler's root directory. Upon execution, profiler connects to the SQL database and retrieves a list of unique md5 hashes corresponding to the various malware samples that have been analyzed by Cuckoo. It then connects to the MongoDB and collects all respective analysis results for each distinct malware. These primary data are then processed and correlated, per sample, to produce a comprehensive malware profile corresponding to the overall behavioral characteristics that were exhibited during each malware's multiple executions. These profiles are then stored in a separate "*profiles*" collection inside the MongoDB. Based on the correlated results, profiler automatically generates a set of human and machine readable reports, in json, txt and html format, as well as various charts that visualize the sample's behavior and activities.

Figure 6.2 demonstrates how profiler collects the primary analysis data and stores the processed results in the "*profiles"* collection as well as in different subfolders inside a "*profiles*" directory.

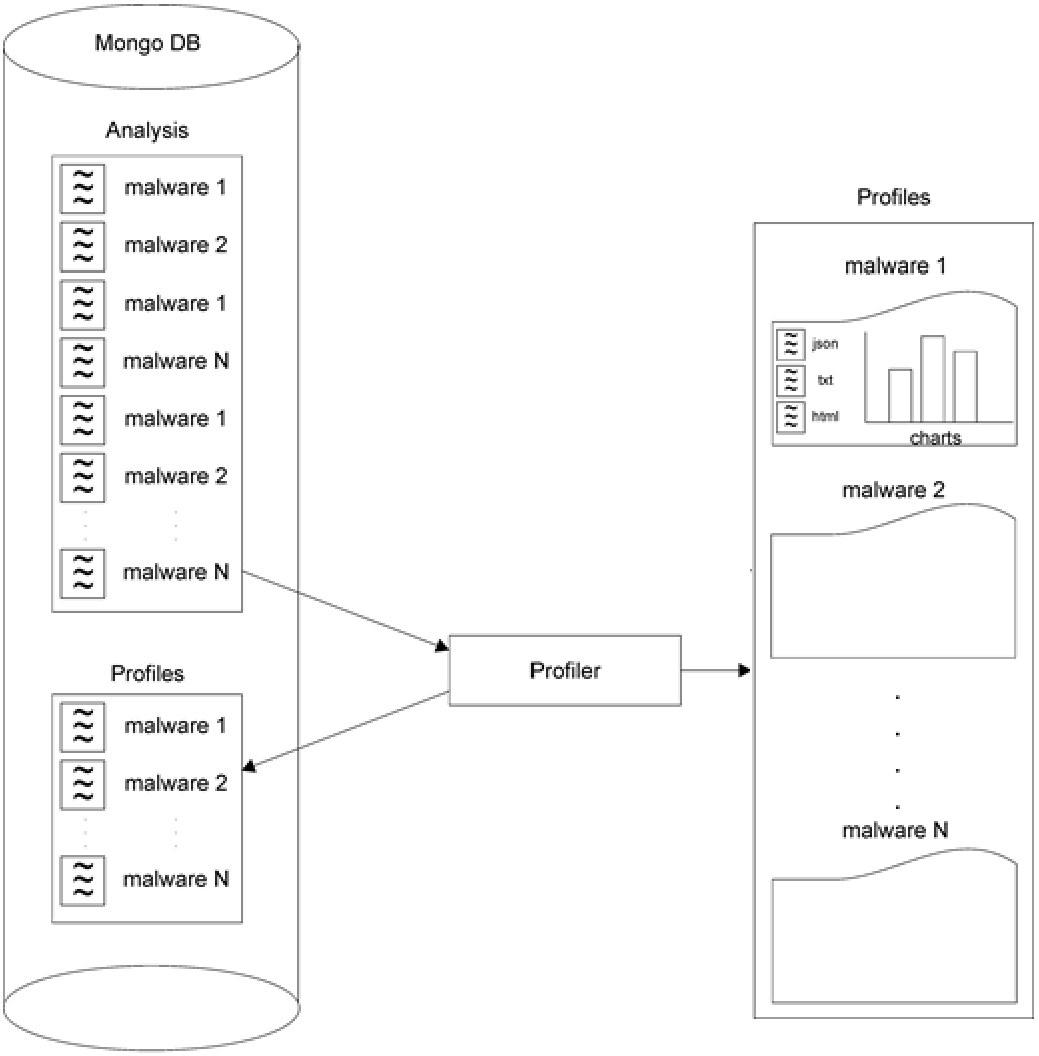


Figure 6.2: Profiler's Collection and Storage procedures.

After the completion of the data processing, profiler automatically terminates. Every time profiler is executed, old malware profiles are updated, upon the recognition of addable analysis results, and new profiles are created for each additional malware sample.

The framework's processing functionality heavily relies on appropriately specifying the exact behavioral characteristics for which intelligence information need to be extracted.

To demonstrate the feasibility of developing a malware forensics framework with the ability to identify multiple behavioral aspects and to asses malware conduct under a specific organizational context, the current implementation focuses on three major malware analysis aspects, without excluding additional information retrieved during the initial analysis process:

- Dropped files.
- Behavioral analysis.
- Network activity.

Profiler has the ability to identify possible similarities and differences in the malware's behavior between multiple executions, relevant to the aforementioned aspects, and can trace any significant behavioral variance back to the specific malware execution that stimulated the observed functionality. To achieve this, a new "*profile*" dictionary structure is used as a reference for comparing and accumulating the multiple analysis results. The new container follows the same structure as the original Cuckoo's results dictionary, with some additional attributes to hold information about any observed difference or similarity to the malware's behavior among its various executions.

The following segment presents profiler's container with the newly added fields highlighted:

```
{
    "info": {
        "started": <timestamp>,
        "ended": <timestamp>,
        "duration": <duration in seconds>,
        "version": <version of Cuckoo>
    },
    "signatures": [
        {
            "severity": <severity level>,
            "description": <signature description>
            "alert": <boolean value>,
            "references": [<any reference link>],
            "data": [<any contextual data>],
            "name": <signature name>
        }
    ],
    "behavior": {
         "processes": [
            {
                "parent_id": <parent PID>,
                "process_name": <process name>,
            "process_id": <PID>,
            "first_seen": <timestamp when the process was first seen>,
            "calls": [
                {
                    "category": <API function category>,
                    "status": <SUCCESS or FAILURE>,
                    "return": <any returned value>,
                    "timestamp": <timestamp of the call>,
                    "repeated": <how many times it was repeated consecutively>,
                    "api": <API function>,
                    "arguments": [
                        {
                            "name": <argument name>,
                            "value": <argument value>
                        }
                    ]
                },
                <...>
            ],
            <...>
        }
    ],
    "processtree": [
        {
            "pid": <PID>,
            "name": <process name>,
            "children": [<recursive child entries>]
        }
    ],
    "summary": {
        "files": [<list of files accessed>],
        "keys": [<list of registry keys accessed>],
        "mutexes": [<list of mutexes accessed>]
    }
},
"static": {<static analysis if available for the file type>},
"dropped": [
    {
        "size": <file size>,
        "sha1": <SHA1 hash>,
        "name": <file name>,
        "type": <file type>,
        "crc32": <CRC32 hash>,
        "ssdeep": <Ssdeep hash>,
        "sha256": <SHA256 hash>,
        "sha512": <SHA512 hash>,
        "md5": <MD5 hash>
    },
    <...>
],
"file": {
    "size": <file size>,
    "sha1": <SHA1 hash>,
    "name": <file name>,
    "type": <file type>,
    "crc32": <CRC32 hash>,
    "ssdeep": <Ssdeep hash>,
    "sha256": <SHA256 hash>,
    "sha512": <SHA512 hash>,
    "md5": <MD5 hash>
},
"debug": {
    "log": <content of analysis.log>
},
"network": {
    "http": [
        {
            "body": <request body>,
            "uri": <request URI>,
            "method": <request method>,
            "host": <host name>,
            "version": <HTTP version>,
            "path": <path of the request>,
            "data": <dump of whole request>,
            "port": <port>
        },
        <...>
    ],
    "udp": [
        {
            "dport": <destination port>,
            "src": <source IP>,
            "dst": <destination IP>,
            "sport": <source port>
        },
        <...>
    ],
    "hosts": [<list of involved IP addresses>],
    "dns": [
        {
            "ip": <IP address>,
            "hostname": <domain name>
        },
    ],
    "tcp": [
        {
            "dport": <destination port>,
            "src": <source IP>,
            "dst": <destination IP>,
            "sport": <source port>
            },
            <...>
        ]
    }
}
```

When a malware sample is processed for the first time, the container starts at an "empty" state. The multiple analysis results of the malware under investigation are compared side by side with the container's data. Dropped files, API calls and Network activities are uniquely inserted inside the dictionary. If a specific characteristic has been exhibited multiple times, a single record is inserted and specific fields are updated with information relevant to the multiple analyses were the specific behavior was detected. If the profile of the given malware has already been created at a previous profiler's execution, the stored data from the MongoDB are passed on to the temporary container and any additional analysis results are compared against the current profile. Therefore the profiles of the malware samples can be created and updated at any point, irrelevant of how often profiler is executed in between the analysis processes of Cuckoo.

The following python code segment demonstrates how dropped files are aggregated and correlated to produce the resulting profile container.

```
if k == "dropped":
        for dropped in res["dropped"]:
            found = False
            found_name = False
            found_size = False
            for i in range(0,len(results_dict["dropped"])):
                if dropped["md5"] == results_dict["dropped"][i]["md5"]:
                    found = True
                    pos = i
                    if dropped["name"] == results_dict["dropped"][i]["name"]:
                        found_name = True
                    if dropped["size"] == results_dict["dropped"][i]["size"]:
                        found_size = True
                    break;
            if not found:
                temp_value = {}
                temp_value = deepcopy(dropped)
                analysis = []
                analysis.append(res["_id"])
                temp_value["analysis"]=analysis
                temp_value["repeated"] = 0
                temp_value["all_execs"] = "No"
```

```
    if len(temp_value["analysis"]) == total:
        temp_value["all_execs"] = "Yes"
    temp_value["same_as"] =[]
    results_dict["dropped"].append(temp_value)
else:
    if res["_id"] not in results_dict["dropped"][pos]["analysis"]:
        results_dict["dropped"][pos]["analysis"].append(res["_id"])
        if not found_name:
            same_as = {}
            same_as["name"] = dropped["name"]
            same_as["in_analysis"] = res["_id"]
            results_dict["dropped"][pos]["same_as"].append(same_as)
    else:
        results_dict["dropped"][pos]["repeated"] += 1
        count_repeated_files += 1
    if len(results_dict["dropped"][pos]["analysis"]) == total:
        results_dict["dropped"][pos]["all_execs"] = "Yes"
        count_dropped_simm +=1
```

Since the name of the files may vary between executions, dropped files are compared based on their MD5 hash value. If the same file was dropped in multiple executions, the respective analysis IDs and MongoDB’s ID are stored in a relevant attribute. Files with different names but with the same MD5 hash are also recorded with details on the specific execution that they appeared. Moreover, if the same file was identified more than once inside the same execution, profiler holds the number of times that it was dropped.

Regarding the behavioral aspects of the malware sample, all processes and API calls are also stored using the same methodology. Information relevant to similar or different processes are recorded. API calls are considered identical if their category, name, status, arguments and repeats are the same. Profiler additionally holds information relevant to API calls with the same name and arguments but different return status and repetitions. The respective python code that processes the API calls can be found in Part I of the Appendix.

Network traffic is processed in the same manner. Information relevant to the specific analysis, in which the same or different http, dns, tcp or udp requests were observed, are respectively stored.

In order to be able to identify and trace a specific behavior back to the original analysis of the sample and the related raw and detailed analysis results, a small modification inside Cuckoo’s processing code took place. Cuckoo currently does not relate the MongoDB’s records with the respective SQL ID of each analysis. Thus it was not possible to connect the mongodb's data with the Cuckoo's detailed analysis results that stored in separate directories for each different analysis request. To this end, we simply customized a small part of a single Cuckoo's processing module so as to additionally store the specific analysis path of the produced results for each analysis request.

All the initial processing results of the above procedure, that is the created profile container for each sample, are stored inside the "profiles" collection of the MongoDB as well as in a generated json report inside a "Profiles" Directory with relevant subfolders named after the malware's MD5 hashes. These files include some initial abstracted and summary data regarding the malware's multiple executions results, as well as additional detailed information on each different behavioral aspect of the given malware.

After the creation of the "profile" container, a new processing procedure takes place in order to analyze the produced results and extract inherent intelligence information relevant to the overall behavior of the examined samples. The outcomes of this analysis are stored in both txt and html formats. Furthermore, in order to visualize the produced results, a number of charts in bar formats are automatically generated graphically presenting information regarding the differences and similarities identified in the dropped files and API calls.

The aforementioned proposed framework refers to utilizing Profiler as an external independent processing tool that correlates and analyzes the results from multiple executions of the same malware sample, producing comprehensive malware profiles. The overall architecture of the proposed framework can be utilized to gain an understanding on how a certain malware behaved under a specific organizational context.

6.2 Integration with Cuckoo

To further enhance the proposed framework's efficiency, usability and overall performance overhead, Profiler can be fully integrated within the operational activities of Cuckoo Sandbox. By appropriately customizing and slightly modifying Profiler's code, the component can be entirely incorporated within Cuckoo's implementation as an extra reporting module. Figure 6.3 presents the modified architecture of the proposed framework, which integrates the modified version of Profiler with Cuckoo.

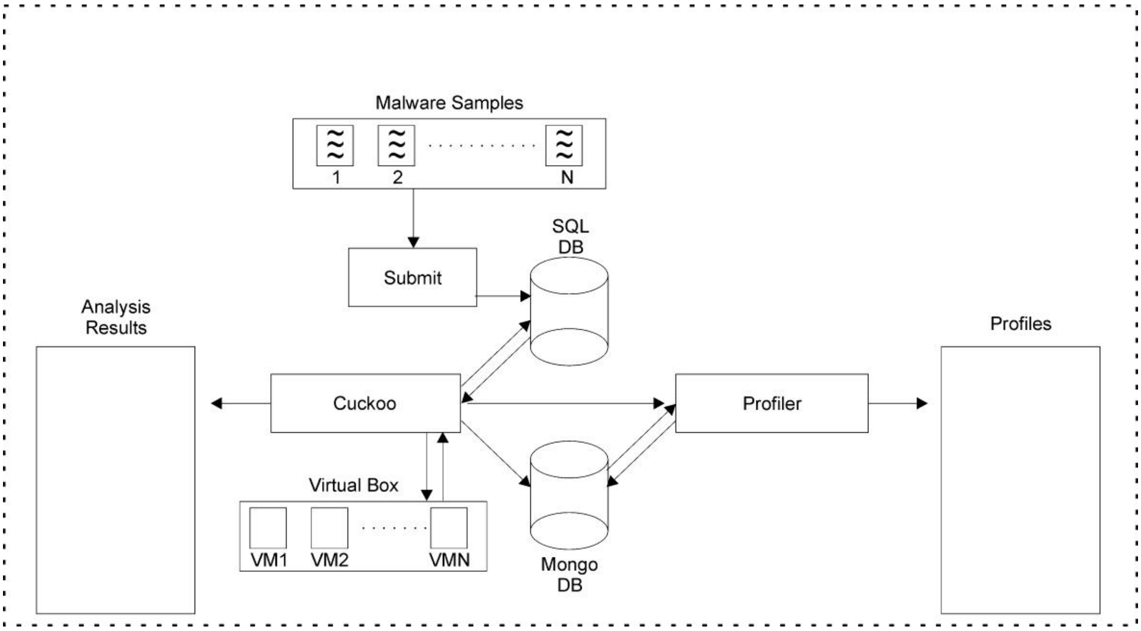


Figure 6.3: Framework’s Integrated Architecture.

The core functionality of the introduced Profiler module remains unaffected. The difference of this approach can be found in the manner that profiler is initiated and executed. Each time Cuckoo performs an analysis on a given sample, a predefined set of reporting modules are called upon, to produce various types of reports. By including profiler as a reporting module, Cuckoo automatically executes Profiler in every malware analysis request. Therefore, the malware’s profiles are automatically created or updated, upon each malware’s execution.

By adopting this specific implementation the overall performance of the proposed framework is significantly enhanced and Profiler’s total analysis time is impressively reduced. This is mostly due to the fact that Profiler performs only one comparison and for a single malware each time it is executed.

6.3 Framework Limitations

Based on the experiments and malware analyses that were conducted in order to asses and evaluate the introduced analysis approach, the proposed malware forensics framework has proven to be efficient and reliable, providing comprehensive results on each malware’s conduct and characteristics. However, the current implementation of profiler and its internal architecture, encapsulate some important limitations with regards to the framework's dependencies with external parameters and possible structural restrictions.

The introduced core processing module of the suggested framework is heavily dependent on the current dictionary structure that Cuckoo uses to store the analysis results. Any future alternations or transformations to the standard dictionary format, could affect the execution of Profiler as it will not be able to recognize the modified structure so as to extract and process the analysis results.

The fact that Profiler utilizes this specific structure also makes it completely Cuckoo dependent. Since Profiler does not process the raw analysis results to produce its own accessible data format, it can only manipulate and operate on the analysis results as they are produced by Cuckoo analyzer.

The limitations regarding the restricted time period in which the framework was developed and realized, also led to restricting various processing aspects of Profiler. The behavioral characteristics of each different malware sample are complex and multivariate. In order to fully investigate all possible aspects and diverse observations, multiple filters and comparisons have to be constructed. Even though the current implementation of profiler can identify and extract many different behavioral features, additional filters and analysis processes would significantly complement the produced malware profiles.

7 Chapter 7 - Conclusions

This final chapter will summarize the work and effort of this research as well as the conclusions and results of our experimentation and analysis methodology.

7.1 Summary

A summary of our research and experimentation methodology and our overall effort and work will be given in this section.

7.2 Contribution

Our contribution will be analyzed and assessed in this section.

7.3 Future Work

Possible future work and research regarding our topic will be discussed in this section.

Bibliography and References

[1]. Von Neumann, John. Theory of Self-Reproducing Automata. In *Essays of Cellular Automata (University of Illinois Press):66-87,* 1966.

[2]. Penrose, Lionel S. “Self-Reproducing Machines”. Scientific American, June 1959: 105-114.

[3]. Stahl, Frederick G. “On Artificial Universes”, 1961. Available at:

http://fredstahl.com/Fred_Stahl/CV_files/On%20Artificial%20Universes.pdf

[4]. Pesavento, U. (1995) An implementation of von Neumann's self-reproducing machine. Artificial Life 2(4):337-354.

[5]. Vyssotsky, Victor A**.** *Darwin: A Game of Survival and (Hopefully) Evolution*. New Jersey: Bell Telephone Laboratories, 1961.

[6]. Dr. Solomon's Virus Encyclopedia, 1995, ISBN 1897661002

[7]. Ralph, Burger. “Computer Viruses: A High Tech Disease”. Abacus, 1988.

[8]. Risak, Veith, "Selbstreproduzierende Automaten mit minimaler Informationsübertragung", Zeitschrift für Maschinenbau und Elektrotechnik, 1972

[9]. Kraus, Jürgen, Selbstreproduktion bei Programmen, February 1980

[10]. McMullin, B. “John von Neumann and the Evolutionary Growth of Complexity: Looking Backwards, Looking Forwards...” Artificial Life 6(4):347-361 by the MIT Press, Fall 2000

[11]. Thomas Chen, Jean-Marc Robert. "The Evolution of Viruses and Worms". 2004

[12]. Bayer, U., Moser, A., Kruegel, C., and Kirda, E. Dynamic analysis of malicious code. Journal in Computer Virology 2, 1, 67–77, 2006

[13]. Moser, A., Kruegel, C., and Kirda, E. 2007. Exploring Multiple Execution Paths for Malware Analysis. In IEEE Symposium on Security and Privacy, Oakland.

[14]. Distler, D. Malware Analysis: An Introduction. SANS Institute, 2007.

[15]. Skoudis, E., Zeltser, L. Malware Fighting Malicious Code. New Jersey: Prentice Hall PTR, 2003.

[16]. Lorna Hutcheson. Malware Analysis The Basics, 2006. Available at:

http://isc.sans.org/presentations/cookie.pdf

[17]. Zeltser, L. Reverse Engineering Malware: Tools and Techniques Hands-On. Bethesda: SANS Institute, 2007.

[18]. Li, Frankie. A Detailed Analysis of an Advanced Persistent Threat Malware. SANS institute, 2011

[19]. Valli, C. & Brand, M. The Malware Analysis Body of Knowledge (MABOK),

Edith Cowan University, School of Computer and Information Science, 2008

[20]. Brand, M., Valli, C. & Woodward, A. Malware Forensics: Discovery of the Intend of Deception. Edith Cowan University, Australian Digital Forensics Conference, 2010.

[21]. Aquilina, J., Casey, E., & Malin, C. Malware Forensics Investigating and Analyzing Malicious Code. Burlington, MA: Syngress, 2008.

[22]. Sikorski, M.,Honig, A. Practical Malware Analysis. No Starch Press. San Francisco, 2012

[23] Egele, M., Scholte, T., Kirda, E., Kruegel, C. A Survey on Automated Dynamic Malware-Analysis Techniques and Tools. ACM Computing Surveys, Vol 44, No 2, Article 6, February 2012

[24] Noah M. Johnson, Juan Caballero, Kevin Zhijie Chen, Stephen McCamant, Pongsin Poosankam, Daniel Reynaud, and Dawn Song. Differential Slicing: Identifying Causal Execution Differences for Security Applications. In Proceedings of the 32nd IEEE Symposium on Security and Privacy, 2011.

[25]. Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, Engin Kirda, and Giovanni Vigna. Efficient Detection of Split Personalities in Malware. In Proceedings of the 17th Annual Network and Distributed System Security Symposium, 2010.

[26]. Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauschek, Christopher Kruegel, and Engin Kirda. Scalable, Behavior-Based Malware Clustering. In Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS), 2009.

[27]. J. Crandall, G. Wassermann, D. Oliveira, Z. Su, F. Wu, and F. Chong. Temporal Search: Detecting Hidden Malware Timebombs with VirtualMachines. In Conference on Architectural Support for Programming Languages and OS, 2006.

[28]. Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Song, D., Yin, H.: Towards automatically identifying trigger-based behavior in malware using symbolic execution and binary analysis. Technical Report CMU-CS-07-105, Carnegie Mellon University School of Computer Science (January 2007).

[29]. Brumley, D., Hartwig, C., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Song, D.: Bitscope: Automatically dissecting malicious binaries. Technical Report CS-07-133, School of Computer Science, Carnegie Mellon University (March 2007)

[30] Dawn Song David Brumley Heng Yin Juan Caballero Ivan Jager Min Gyung Kang Zhenkai Liang James Newsome Pongsin Poosankam and Prateek Saxena. Bitblaze: A new approach to computer security via binary analysis. 2008.

[31]. Microsoft Security Intelligence Report, Volume 11. An in-depth perspective on software vulnerabilities and exploits, malicious code threats, and potentially unwanted software in the first half of 2011. Microsoft Corporation, 2011.

Available at: http://www.microsoft.com/security/sir/archive/default.aspx

[32]. Brand, M. Forensic Analysis Avoidance Techniques of Malware. Paper presented at the 5th Australian Digital Forensics Conference, Edith Cowan University, Mount Lawley Campus, Western Australia, 2007.

[33]. M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In Proceedings of the 12th USENIX Security Symposium (Security'03), pages 169–186, Washington, DC, USA, Aug. 4–8, 2003. USENIX Association.

[34]. SHARIF, M., LANZI, A., GIFFIN, J., AND LEE, W. Impeding malware analysis using conditional code obfuscation. In Proceedings of the 15th Annual Network and Distributed System Security Symposium, 2008.

[35]. C. Linn and S. Debray. Obfuscation of Executable Code to Improve Resistance to Static Disassembly. In ACM Conference on Computer and Communications Security, 2003.

[36]. G.Wroblewski. General Method of Program Code Obfuscation. PhD thesis, Wroclaw University of Technology, 2002.

[37]. P. Szor. The Art of Computer Virus Research and Defense. Addison Wesley, 2005.

[38]. Weber, L. Dynamic Analysis of Malware. Master Seminar, Horst-Görtz Institute 2010.

[39]. Chen, X., Andersen, J., Mao, Z.M., Bailey, M., Nazario, J.: Towards an Understanding of Anti-Virtualization and Anti-Debugging Behavior in Modern Malware. In: Proceedings of the 38th Annual IEEE International Conference on Dependable Systems and Networks, 2008.

[40]. Kang, M.G., Yin, H., Hanna, S., McCamant, S., Song, D.: Emulating Emulatio-Resistant Malware. In: Proceedings of the 2nd Workshop on Virtual Machine Security, 2009.

[41]. Simson Garfinkel, Alex J. Nelson and Joel Young. A general strategy for differential forensic analysis. Digital Investigation 9, S50–S59. Published by Elsevier Ltd. 2012.

[42]. Gursimran Kaur and Bharti Nagpal. Malware Analysis & its Application to

Digital Forensic. International Journal on Computer Science and Engineering (IJCSE), Vol. 4 No. 04. 2012.

[43]. Kris Kendal. Practical Malware Analysis. Mandiant- Intelligent Information Security, 2007.

[44]. Dimitris Iliopoulos, Christoph Adami and Peter Szor, Darwin inside the machines: malware evolution and the consequences for computer security, Virus Bulletin Conference, 2008.

[45]. Qing Li, Chris Larsen, Tim van der Horst, "IPv6 - a Catalyst and an Evasion Tool for Botnets and Malware Distribution Networks," Computer, 10 Sept. 2012. IEEE computer Society Digital Library. IEEE Computer Society.

[46]. Antonios Atlasis, Attacking ipv6 implementation using fragmentation. http://media.blackhat.com/bh-eu-12/Atlasis/bh-eu-12-Atlasis-Attacking_IPv6-WP.pdf, March 2012.

[47]. Fernando Gont, “Results of a Security Assessment of the Internet Protocol version 6 (IPv6)”. Research project carried out on behalf of the United Kingdom’s Centre for the Protection of National Infrastructure. Presentation available at:

http://www.si6networks.com/presentations/hacklu2011/fgont-hacklu2011-ipv6-security.pdf, Sept. 2011.

[48]. Marc Heuse, “Recent advances in IPv6 insecurities”. 27th Chaos Communication Congress, 2010.

[49]. Frederick. Cohen, "Computer viruses: theory and experiments,", 1984. Available at: http://web.eecs.umich.edu/~aprakash/eecs588/handouts/cohen-viruses.html

[50]. Frederick. Cohen, “A Short Course on Computer Viruses”. ASP Press, 1990.

[51]. Avinash Kak, “Malware: Viruses and Worms”. Lecture Notes on “Computer and Network Security”, Purdue University, 2012.

[52]. David Harley, Robert Slade, Urs Gattiker, “Viruses Revealed”. 2001

[53]. Spafford, E. H. “The Internet worm incident”. In Proceedings of the 2nd European Software Engineering Conference. 446–468. 1989.

[54]. Cuckoo Sandbox. Automated Malware Analysis. Available at: http://www.cuckoosandbox.org/

[55]. Vishrut, Sharma. “An Analytical Survey of Recent Worm Attacks”. In IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.11, November 2011.

[56]. Murray, Brand. “Analysis Avoidance Techniques of Malicious Software”. Edith Cowan University, Perth, WA. November 2010.

[57]. Glenn, Gebhart. “Worm Propagation and Countermeasures”. SANS Institute 2004.

[58]. Bit Defender. Malware History. Available at :

http://download.bitdefender.com/resources/files/Main/file/Malware_History.pdf

[59]. J. Shoch, J. Hupp, "The 'worm' programs - early experience with a distributed computation," Commun. Of ACM, vol. 25 , pp. 172-180, March 1982.

[60]. G. Smith, “The Virus Creation Labs: A Journey into the Underground”, American Eagle Publications, Tucson, AZ, 1994.

[61]. Botnets The Killer Web Applications.

[62]. H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, L. Wang. On the Analysis of the Zeus Botnet Crimeware Toolkit. Eighth Annual International Conference on Privacy Security and Trust (PST), 2010.

[63]. Yuri, Namestnikov. The economics of botnets. Kaspersky Lab, 2009.

[64]. Johannes. Bauer, Michel. van Eeten, John. Groenewegen, Wolter. Lemstra. “The Economics of Malware. Problem Description, Literature Review and Preliminary Research Design”. Report to the OECD Working Party on Information Security and Privacy and the Ministry of Economic Affairs of the Netherlands. 2007.

[65]. Samuel C. McQuade, III. “Encyclopedia of Cybercrime”. Greenwood Press, 2009.

[66]. Ilsun. You, Kangbin. Yim, “Malware Obfuscation Techniques: A Brief Survey”. International Conference on Broadband, Wireless Computing, Communication and Applications, 2010.

[67]. Thomas. M. Chen, Gregg. W. Tally, “Network Worms”. Encyclopedia of Information Science and Technology, 2nd Edition, pages 2783-2788, 2009.

[68]. Vishrut. Sharma, “An Analytical Survey of Recent Worm Attacks”. IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.11, November 2011.

[69]. L. Wenke, W. Cliff, and D. David, Eds., Botnet Detection: Countering the Largest Security Threat, ser. Advances in Information Security. Springer-Verlag New York, 2008, vol. 36.

[70]. Aleksandr Matrosov, Eugene Rodionov, David Harley, and Juraj Malcho, “Stuxnet Under the Microscope”, Revision 1.2, ESET, November, 2010.

[71]. OECD, Organization for Economic Co-operation and Development. “Malicious Software (Malware): A Security Threat to the Internet Economy”. Report presented at the OECD Ministerial Meeting on the Future of the Internet Economy, June 2008.

[72]. Monirul Sharif, Andrea Lanzi, Jonathon Giffin, and Wenke Lee. Automatic reverse engineering of malware emulators. In Proceedings of the IEEE Symposium on Security and Privacy, 2009.

[73]. Brand, M., Valli, C., Woodward, A. Lessons Learned from an Investigation into the Analysis Avoidance Techniques of Malicious Software, 2010.

[74]. Moser, A., Kruegel, C., and Kirda, E. 2007. Limits of static analysis for malware detection. In 23rd Annual Computer Security Applications Conference (ACSAC). 421–430.

[75]. Royal, P., Halpin, M., Dagon, D., Edmonds, R., Lee, W.: Polyunpack: Automating the hidden-code extraction of unpack-executing malware. In: Jesshope, C., Egan, C. (eds.) ACSAC 2006. LNCS, vol. 4186. Springer, Heidelberg (2006).

[76]. Martignoni, L., Christodorescu, M., Jha, S.: Omniunpack: Fast, generic, and safe unpacking of malware. In: Choi, L., Paek, Y., Cho, S. (eds.) ACSAC 2007. LNCS, vol. 4697. Springer, Heidelberg (2007).

[77]. Kang, M.G., Poosankam, P., Yin, H.: Renovo: a hidden code extractor for packed executables. In: Proceedings of WORM (2007)

[78]. M. Sharif, V. Yegneswaran and H. Saidi, et al. Eureka: A Framework for Enabling Static Malware Analysis. In Proc. of ESORICS'08, 481-500, 2008.

[79]. G. Wagener, R. State, and A. Dulaunoy. Malware Behaviour Analysis. Journal in Computer Virology, 4:279–287, 2008.

[80]. Lorenzo Martignoni, Roberto Paleari, and Danilo Bruschi. A Framework for Behavior-Based Malware Analysis in the Cloud. In Proceedings of the 5th International Conference on Information Systems Security (ICISS), 2009.

[81]. A. Vasudevan and R. Yerraballi. Cobra: Fine-grained Malware Analysis using Stealth Localized Executions. In Proceedings of the IEEE Symposium on Security and Privacy, 2006.

[82]. A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: Malware Analysis via Hardware Virtualization Extensions. In Proceedings of the ACM Conference on Computer and Communications Security (CCS), 2008.

[83]. U. Bayer, E. Kirda, and C. Kruegel. Improving the efficiency of dynamic malware analysis. In Proceedings of the 2010 ACM Symposium on Applied Computing, pages 1871–1878. ACM, 2010.

[84]. Jacob, G., Debar, H., Filiol, E.: Behavioral detection of malware: from a survey towards an established taxonomy. Journal in Computer Virology 4(3) (2008).

[85]. PaiMei - a reverse engineering framework. Available at:

http://github.com/OpenRCE/paimei

[86]. Anubis. Analysis of unknown binaries. http://anubis.iseclab.org.

[87]. Bellard, F. 2005. QEMU, a Fast and Portable Dynamic Translator. In FREENIX Track of the USENIX Annual Technical Conference.

[88]. Willems, C., Holz, T., and Freiling, F. 2007. Toward automated dynamic malware analysis using CWSandbox. IEEE Security and Privacy 5, 2, 32–39.

[89]. Virtual Box. Available at : https://www.virtualbox.org/

[90]. Michael Ligh, Steven Adair, Blake Hartstein and Matthew Richard. Malware Analyst's Cookbook and DVD. Wiley Publishing, Inc., 2011.

[91]. K. Rieck, P. Trinius, C. Willems, and T. Holz. Automatic Analysis of Malware Behavior using Machine Learning. In Journal of Computer Security, 2011.

[92]. Nicolas Falliere, Liam O Murchu, Eric Chien. W32.Stuxnet Dossier, Symantec Security Response. 2011.

[93] H.Wang, S.Jha, and V.Ganapathy. Netspy: Automatic generation of spyware signatures for NIDS. In Proceedings of Annual Computer Security Applications Conference, 2006.

Appendix