University College London

THE MALWARE SCHISM: INFERRING MALWARE TYPES FROM NETWORK TRAFFIC

Aristeidis Athanasiou

Supervisor: Gianluca Stringhini

MSc. Information Security

This report is submitted as part requirement for the MSc in Information Security at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

2 0 1 5

Abstract

Malicious software has been an ongoing threat to computer systems and networks for the past decades. The dawn of the 21st century has seen a proliferation in the numbers, complexity and viciousness of malware. The efforts undertaken by the scientific community in dealing with this persistent threat have been repeatedly proven to be inadequate. Most of the current detection techniques rely on signatures which malware authors can trivially evade. In addition, the vast amount of malware detected daily has rendered detection and classification by manual analysis practically infeasible. Therefore, new, more sophisticated approaches are required that would minimise human interaction and would allow for fast processing of large numbers of files.

In this thesis, we present a novel system capable of automatically classifying malware based on the network traffic they produce by applying machine learning techniques. Our system consists of two modules, the first automatically infers the spreading mechanism of the samples under analysis, while the latter allows for the efficient classification of malicious programs according to their behaviour. Samples that do not fit in any of the existing classes would act as an "alert" about new cyber-criminal activity or a new attack vector. Furthermore, an automated classification of the malware detected in a network would allow system and network administrators to prioritise the mitigation of threats according to their severity.

Keywords: Malware, Dynamic analysis, Traffic analysis, Machine Learning

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Forty five years since the appearance of the first malicious program; the threat of malicious software (malware) is greater than ever before. For the past decades there has been an ongoing arms race between computer security experts (white hats) and cyber-criminals (black hats). Security experts and anti-malware vendors design and implement systems for the detection, classification and mitigation of malware. On the other hand, malware authors have been systematically responding by improving the sophistication of their code, producing malware that evades most detection techniques. Unfortunately, cyber-criminals are winning this digital war, as they have been successful in coming up with ways to intelligently bypass the security mechanisms and avoid detection.

The efforts of the scientific community and anti-virus (AV) vendors are thwarted by the massive number of new malicious programs created on a daily basis. In particular, according to a report published by Panda Security for the first quarter of 2015, the company every day receives the astonishing number of $225,000$ malware samples [1]. Clearly, these numbers render manual analysis practically impossible. Anti-virus vendors need automated systems capable of efficiently differentiating between samples that require manual analysis and those that constitute variants of known threats. Malware identification and classification are of equal importance to detection, as they enable the development of anti-malware tools, assist in orchestrating mitigation and facilitate in devising disinfection mechanisms. Finally an automated system capable of efficiently classifying large numbers of samples, would allow researchers to prioritise their efforts according to the severity of the threat.

1.1 Contributions

The major contributions of this work are:

- We devised a malware taxonomy that explicitly separates malware types referring to propagation methods and self-containment, from types describing the behaviour and purpose of a malicious program.

- We implemented a system capable of classifying malware samples by inspecting network traffic and applying machine learning techniques. The first component of the system infers the spreading mechanism of the underlying samples, while the second classifies the specimens under analysis with respect to their behaviour. Compared to similar approaches, in which a limited number of distinct families was used, our data set consists of over 30 distinct families, each with several variants.
- Our spreading mechanism classifier was evaluated using $7,201$ samples and yielded 95.1% true positive and 4% false positive rates.
- The behaviour-based classification component of our system was tested using $3,943$ samples and achieved 91.4% true positive and 5% false positive rates.

1.2 Structure of this Report

The rest of this report is organised as follows:

- Chapter 2 provides a high-level description of the different types of malicious software. We believe understanding the behavioural differences between the distinct malware types is essential to apprehend how they can be leveraged to assist the classification process. In addition, understanding the intrinsic characteristics of the different malware types is necessary to conceive the capabilities and limitations of the system introduced.
- In chapter 3 we review some of the most common approaches for malware detection and classification. The aim of this chapter is to present the inherent limitations of conventional approaches and familiarise readers with the techniques used by malware authors to elude detection.
- In chapter 4 we present some state-of-the-art malware detection systems that go beyond the traditional techniques and overcome many of their limitations. In particular, we focus on systems that rely on traffic analysis and machine learning. The goal of this chapter is to provide the intuition behind the system developed.
- Chapter 5 provides the reader with an overview of the samples included in our data set. An awareness of the characteristics of our data set assists the reader in understanding the scope of this work.
- Chapter 6 presents the system implemented in the context of this study. In particular, we provide a detailed description of our system and discuss the approaches taken in establishing a ground truth and devising the features used by our classifier. In addition, we present a thorough analysis and discussion of our system's results.
- Finally, chapter 7 presents a summary of our findings and suggests potential improvements and extensions of our system.

Chapter 2

Malware Types

This chapter provides the reader with pertinent background information, necessary for understanding the rest of this work. The discussion centers on understanding the heterogeneity of malicious programs and their intrinsic characteristics. In addition, it illustrates how writing malicious code has evolved from a form of digital vandalism to an underground economy. The rest of the chapter is organised as follows:

- Section 2.1 presents the milestones of malware evolution from 1984 to 2010. It is our firm belief that, a timeline of the most notorious malicious programs can assist the reader in understanding how malware sophistication and prevalence have escalated throughout the past decades.
- In section 2.2 we provide a taxonomy of malware types explicitly segregating those referring to a program's spreading-mechanism and self-containment from those describing its underlying behaviour and purpose.

2.1 Malware Timeline

In this section, we discuss the milestones of malware evolution, from 1948 when Von Neuman set the theoretical foundations of malware, to 2010 when Stuxnet, a malicious program with unprecedented complexity, struck an Iranian nuclear plant.

In 1948, John Von Neuman started giving lectures on "Self-reproducing automata" at the University of Illinois. Eighteen years later, Arthur W. Burk compiled these lectures and published a paper illustrating the potential of computer programs to replicate themselves. Despite the fact that at the time the implementation details were not conceivable, it is regarded as the theoretical work on which computer viruses are based upon.

In 1971, Bob Thomas created the Creeper[1] as part of an attempt in verifying the theories of Von Neuman. Even though, the term "virus" did not exist in this context at the time,

[1] `http://virus.wikidot.com/creeper`

the "Creeper" is considered to be the first computer virus.

The year 1975 the first Trojan horse made its debut. The Pervading Animal [2] kept computer users busy with a guessing game, while at the same time copied itself to every directory the user had access to.

Six years later, the program Elk Cloner[2], developed by a young student, infected Apple II systems through floppy disks and was responsible for the first large-scale virus outbreak.

During the same year, Jon Hepps and John Shock developed the first worms[3] while working at the Xerox Palo Alto Research Center. The programs would independently travel through the network carrying out useful tasks such as posting announcements and improving the utilization of systems. The potential impact worms could inflict on computer systems became apparent, when a worm started malfunctioning and crashed all the computers of the research center.

In 1983 Fred Cohen established the term "virus" to describe programs that can self-replicate by infecting other programs using an analog to biological viruses [3].

Three years later, Basit Farooq Alvi and Amjad Farooq Alvi built Brain[4], the first IBM-PC compatible virus. Brain spread via floppy disks and infected the boot sector virus of the DOS operating system.

In 1988, Robert Tappan Morris, created the Internet worm, which was the first of its kind, spreading using the modern internet. Although Morris' intent was not malicious, a bug in the code caused the program to spread uncontrollably [4]. As a consequence, approximately $5,000$ systems were infected, the internet had to be shut down and the damages were estimated up to $\$10,000,000$.

In 1990, Mark Washburn and Ralf Burger after analysing the Vienna and Cascade viruses created Chameleon, which is accepted as the first polymorphic virus. Chameleon used encryption as well as obfuscation techniques to create modified versions of itself.

Five years later, McNamara crated the first macro virus using the macro language Word-BASIC. Concept is accepted as the first multi-environment virus as it was capable of infecting both Windows and Mac OS via Microsoft Word documents. The first Excel macro virus, Laroux, arrived one year later.

In 1999 David L. Smith was sentenced to 10 years of imprisonment for creating the Mellisa macro virus. Melissa is regarded as the first mass-mailing macro virus sending

[2] http://www.skrenta.com/cloner/
[3] http://virus.wikia.com/wiki/Xerparc
[4] http://campaigns.f-secure.com/brain/virus.html

unsolicited mails to the first 50 contacts in the victim's address book. The fearsome malware infected about 1/5th of the computers worldwide.

The first year of the new millennium, tens of millions computer users received an email with the subject line "ILOVEYOU". Upon execution of the attached file "LOVE-LETTER-FOR-YOU.txt.vbs" the worm damaged the victim's computer and sent emails to all the contacts saved in the Windows Address Book.

The following year, Code Red exploited a known buffer-overflow vulnerability[5], becoming the first worm using exploit-based propagation. The worm was designed to perform denial of service attacks against a predefined list of websites, including the web server of the White House.

In 2002, Beast, one of the first backdoor Trojan horses made its debut. Beast gained popularity among cyber-criminals as a Remote-Administration-Tool (RAT), granting full control of the infected machine to the attacker and providing innovative features.

The following year, the SQL Slammer worm achieved an unprecedented spreading rate[6], doubling in size every 8.5 seconds and infecting $75,000$ computers within 10 minutes. The worm exploited a disclosed vulnerability in Microsoft's SQL Server and although it did not contain malicious payload, it created havoc by disrupting networks and taking down database servers [5].

In 2007, the Storm worm started spreading via emails with a subject line about a disaster caused by a storm, hence its name. The infected hosts would become part of a peer-to-peer (P2P) botnet whose total computing power was estimated to be comparable with that of the supercomputers of the time [6].

Two years later, Conficker started its propagation by exploiting a vulnerability[7] found in several versions of Windows. The estimated number of compromised hosts ranged between 9 and 15 million systems, forming one of largest botnets in history [7].

The last malware of this short historical overview is Stuxnet, which was created in 2010. Stuxnet used 4 zero-day exploits[8], was capable of infecting air-gaped machines via USB and its sophistication was considered groundbreaking. In particular, Stuxnet contained a high specialised payload targeting the SCADA systems of Iranian nuclear facilities [8]. Figure 2.1 portrays the evolution of malicious programs between 1980 and 2010.

[5] http://technet.microsoft.com/library/security/ms01-033
[6] http://www.caida.org/publications/papers/2003/sapphire/sapphire.html
[7] https://technet.microsoft.com/library/security/ms08-067
[8] http://www.zdnet.com/article/stuxnet-attackers-used-4-windows-zero-day-exploits/

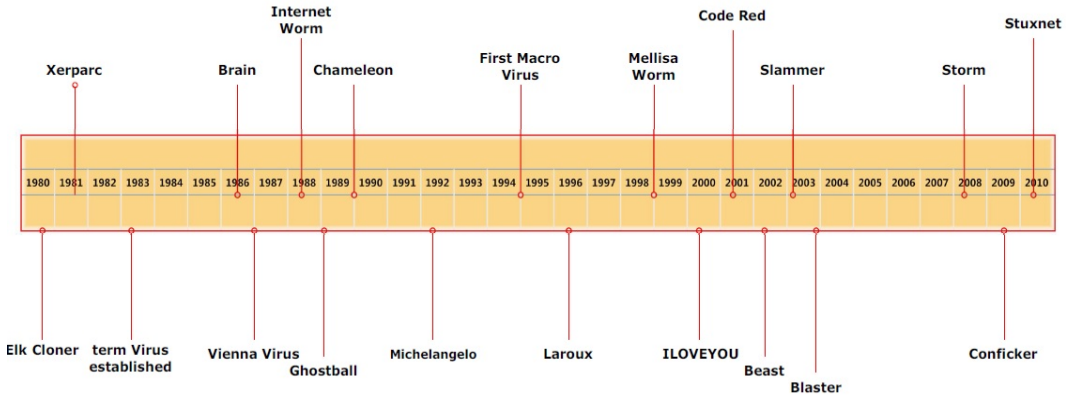

Figure 2.1: Milestones in malware evolution (1980-2010).

2.2 Different Flavours

In 2009 S. Kramer and J. C. Bradfield defined malware as "a software system that damages non-damaging software systems or software systems that damage malware" [9]. We believe this definition does not capture the heterogeneity, variety and purpose of all modern malware. Therefore, in this paper we define malware as software that is executed in ignorance of the user performing some unwanted activity. We highlight that this catchall definition is not capable of illustrating the diversity of modern malware and the segregation of the different malware types is essential. We devised a taxonomy grouping malware types in two ways; the first is in terms of their self-containment and spreading mechanism and the latter is based on their behaviour and purpose.

In 2.2.1 we provide a taxonomy based on the spreading mechanism and whether or not the existence of a host program is a prerequisite. At the same time, we acknowledge the inherent limitations of any malware taxonomy because of the obscure bounding lines between the different types. Next, we continue by presenting a list of the different malware types, grouped based on their behaviour and purpose in 2.2.2. We stress that the different categories of malware are by no means mutually exclusive and it is very common for a malicious program to have more than one attack vectors. We believe comprehension of the distinctive features of different types is of paramount importance in understanding the scope and limitations of the system introduced.

2.2.1 Spreading and Self-containment based Taxonomy

In the following section we adduce a malware taxonomy based on the spreading mechanism and the necessity for a host program; Figure 2.2 illustrates the corresponding Venn diagram.

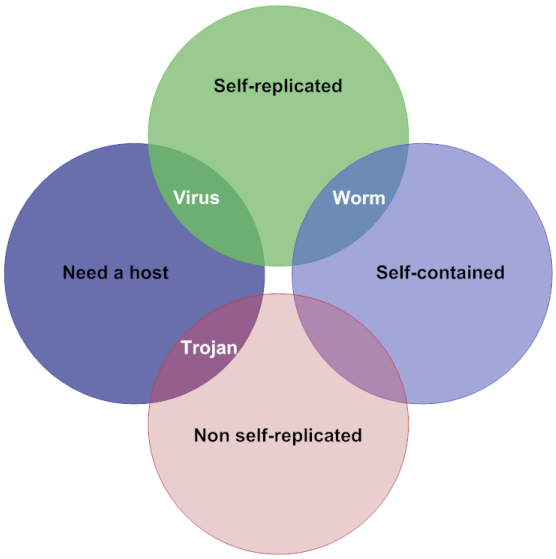

Figure 2.2: Venn diagram representing the different malware types in terms of spreading mechanism and self-containment.

Viruses

In 1984, Fred Cohen coined the term "virus", according to which, a virus is defined as "a computer program that can infect other computer programs by modifying them to include a copy of itself" [3].

Similarly to their biological counterpart, computer viruses are incapable of surviving on their own and require a host program [10]. Their most striking feature is their replication capability, which allows them to modify other programs by getting attached to them and infect new systems. In addition, computer viruses can mutate and evolve during the replication process by attaching an improved version of themselves to the victim file, thus improving their survivability.

Once the infected file is opened, the malicious code gets automatically executed. Typically, once a virus infects a system, it performs some kind of malicious activity such as corrupting files or affecting the system's performance. However, there also exist viruses that do not perform any specific activities in order to remain as stealthy as possible and avoid detection for the sake of future exploitation.

Our hypothesis involves viruses producing substantially less traffic compared to worms and Trojans. In particular, we estimate that since viruses rely on file infection for their propagation rather than utilising the network, they generate limited network traffic. However, we expect that a non-negligible proportion of viruses renders the victim part of botnet, and consequently produce a substantial amount of traffic.

Trojan Horses

The name of this malware family stems from the wooden horse Ancient Greeks used to infiltrate Troy after a ten year siege. Trojan horses or Trojans, are defined as programs that masquerade as useful, while in the background perform unauthorised actions [11]. The malicious payload of a Trojan horse is either contained inside the seemingly beneficial program or is downloaded later on. Unlike viruses and worms, Trojans do not self-replicate, and their installation requires the explicit user consent. Trojans span a very large attack vector and consequently a behaviour and purpose-based classification is more appropriate.

Worms

Similarly to viruses, worms are capable of replicating themselves by infecting hosts that are within reach of the (initial) victim. The subsequent replicas have self-replication capabilities too; this results to an exponential spreading rate and an "avalanche" effect. The infection and propagation model of a computer worm shares many similarities with the proliferation of biological pathogens during the infection of a living organism [12, 13]. Unlike computer viruses, worms are self-contained programs and can exist independently from any other program. Worms can be separated into two large categories, depending on whether they require human interaction or not.

Autonomous worms: Worms belonging to this category spread by exploiting computer vulnerabilities or system and network misconfigurations. Typically, soon after the infection of a victim, they seek their new targets by port-scanning local and remote hosts in order to find and exploit vulnerable machines. In comparison with worms that require human interaction their spreading rate is much higher.

Worms involving user: This category mainly consists of worms using email harvesting. Examples include the infamous ILOVEYOU [14] and Mellisa as well as worms spreading via infected Office documents. Once the victim opens a malicious file attached to an email, the worm sends a replicate of itself to the contacts stored in the victim's address book. Over the past years, the prevalence of this type of worms has seen a radical decrease since all major email providers scan email attachments for potential threats.

It is very common for worms to probabilistically compute the IP address that they will attempt to scan and attack next. However, there are certain special purpose blocks of IP addresses that are reserved by the Internet Assigned Numbers Authority (IANA[9]) [15]. Since a worm typically tries to attack as many hosts as possible by scanning arbitrary IP addresses, it is likely to visit an address belonging to a reserved block. Considering that a user is highly unlikely to visit one of the reserved addresses, the existence of network traffic towards them, would be a strong indication of a worm infection. Another characteristic of worms, is that they connect to a vast number of hosts within few

[9] http://www.iana.org/

minutes or even seconds. In addition, usually the hosts visited are completely unrelated to each other and spread around the world, which is not in accordance with the typical behaviour of the internet user. Finally, the profile of the communication between a worm-infected host and potential victims significantly differs from the profiles of typical internet communications in terms of the duration and number of packets exchanged.

2.2.2 Behaviour and Purpose-based Listing

In this section we discuss different malware behaviours; we emphasise that the underlying categories are not conflicting and is not uncommon for malicious programs to combine several of the characteristics described below. Awareness of the different malware functions and goals is key in understanding the advantages and limitations of a system that relies on traffic analysis, like the one introduced. Figure 2.3 illustrates the different behaviours and purposes.

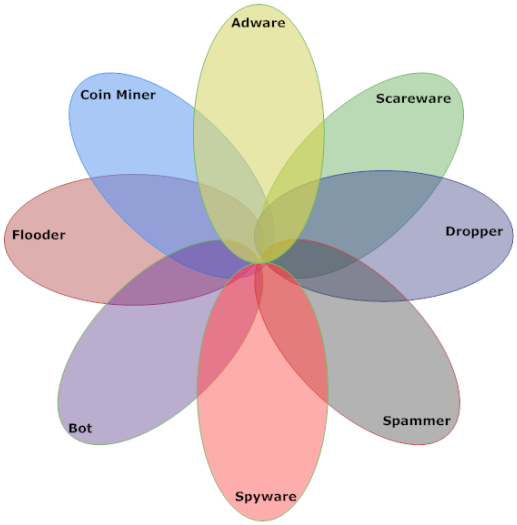

Figure 2.3: The different malware behaviours and purposes.

Adware

According to the Anti-Spyware Coaliton[10], *adware* is defined as "advertising display software that delivers advertising content potentially in a manner or context that may be unexpected and unwanted by users". Adware is also referred as potentially unwanted programs (PUPs), however there exists a fine line between the two categories. Adware tracks the victim's browsing habits and collect data from the infected system without the user's consent, in order to present them with targeted advertisements, typically in the form of banners or pop-up windows. Others, hijack the victim's browser start page or default search engine and redirect them to adverting web pages.

[10] `http://www.antispywarecoalition.org/`

One key difference between adware and PUPs, is that the latter are legitimate. The proliferation of internet targeted advertising [16] has led many companies to distribute their software for free (freeware) and rely on the advertising functions embedded in them for their revenues. Moreover, many web sites including CNET[11], Brothersoft[12] and Softonic[13] have custom installers for commercial and open-source programs that install PUPs along with the desired software [17]. When a user installs a program using one of the aforementioned installers, he agrees with the included End User License Agreement (EULA) giving his consent to install the unwanted program.

Our system regards adware and PUPs as one family, in particular, we use the term adware as an umbrella term that encompasses the behaviour and purpose of both. Intuitively, adware programs contact advertising servers to render customised advertisements or redirect user queries to advertising web pages. Our system takes advantage of that, by monitoring queries to such servers and classifying the underlying samples as adware.

Rootkits

The most conspicuous feature of *rootkits* is their ability to hide inside the victim's operating system in order to cloak their presence and grant the attacker privileged access to the system. Typically, their role is to conceal the existence of other malicious programs operating in the system, by subverting operations that would reveal their existence [18]. Rootkits are typically installed by the attacker as part of a blended threat, as discussed in 2.2.3. Although rootkits are strongly associated with cyber-criminals, they have been also used by reputable companies as a means of gathering information from their clients. One of the most eye-opening events was the "Sony BMG rootkit scandal[14]" when Sony distributed 22 million music CDs containing code for enforcing digital rights management (DRM). Classification of malicious files as rootkits is not included among the aims of our system, since rootkits tend to be very stealthy and are not expected to produce any network traffic at all.

Scareware

Scareware refers to malicious programs that attempt to terrorise computer users, in order to extort payment. Some of the most notorious Scareware include *Ransomware* and fake anti-virus programs. Ransomware involves some form of extortion, and it can be further separated depending on its ability to damage the target system or not.

Ransomware with destructive capabilities uses public key cryptography to encrypt the victim's file system and demand a fee in exchange for the decryption key. As its name

[11] http://download.cnet.com/windows/
[12] http://www.brothersoft.com/
[13] http://en.softonic.com/
[14] https://www.schneier.com/blog/archives/2005/11/sony_secretly_i_1.html

suggests, they hold the compromised system hostage and threat their victims with permanent loss of their data, unless the ransom fee is deposited within a short deadline (cryptoviral extortion) [19].

Others, less damaging ones, present the victim with a forged page that pretends to originate from some law enforcement authority informing the victim that he has committed some law violation. Typically, the unlucky victim is informed that his computer is housing child pornography or that he has violated international copyright laws and that unless he deposits the necessary fee, he will be prosecuted.

Spyware

Spyware constitutes a major threat to computer users' privacy and personal data. It scans the victim's system collecting stored data and monitor information being processed in real time. Spyware is capable of collecting all kinds of information including stored cookies and credentials, credit card numbers and keystrokes, which can be later used to impersonate the victim to an application or a web page. Spyware programs vary in terms of how intrusive to the victim's privacy they are. In particular, the most invasive ones allow the attacker to even capture live video and audio from the computer's camera.

Droppers

A *dropper* is typically used at an early stage of an attack to install additional malware to the target system. The additional malicious payload is either compressed and encapsulated inside the dropper, or it is downloaded through the internet. Once a dropper gets executed, it loads the malicious code into the memory of the compromised machine. Intuitively, a dropper downloading additional malware produces significant traffic. In particular, the bytes downloaded from a remote host are expected to overwhelm the bytes sent; this can be leveraged to classify a malicious program as dropper.

Coin Miners

This category describes malware that uses the victim machine to mine cryptocurrencies. Undoubtedly, Bitcoin [20] is the most popular cryptocurrency, and therefore constitutes the main target of these programs. Coin miners can significantly hinder the performance of the infected system, as harvesting Bitcoin is computationally expensive. This malware category is far from stealthy, as the significant system slowdowns it induces, can not be overseen. We estimate that coin miners produce limited traffic and they could be easier detected by inspecting system logs rather than network traffic.

Flooders

Malware falling in this class perform denial of service (DoS) attacks against web pages and web services by sending a tremendous amount of traffic. Typically *Flooders* utilise the Hypertext Transfer Protocol (HTTP) to flood victim sites with multiple requests.

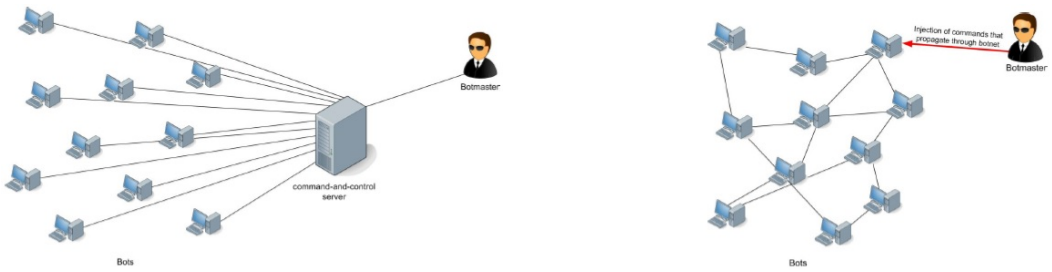

Figure 2.4: Centralised botnet (left) and P2P botnet (right), source [21].

We estimate that the majority of the requests sent, use the POST method rather than the GET. This is because, the POST method allows for form or even file submission via HTTP. The additional computational cost required for processing the received data, renders the attack very effective. The victim host is overwhelmed with a substantial volume of data, that is not able to process fast enough and its resources are depleted.

Bots and Botnets

The name *bot* is an abbreviation of the word robot and refers to the way the compromised machines do the bidding of their master. Bots are presumably the most fearsome malware, because the attacker can remotely control the compromised machines by establishing with them a command and control (C&C) channel. When a machine gets infected by the malicious program, it is turned into a *zombie*, blindly following the directions of the cyber-criminal behind the attack. Machines that are compromised by the same payload and are controlled by the same person or team form a *botnet*. The individual responsible for controlling, maintaining and issuing the commands to the infected machines is reffed as the *botmaster*.

The commander orchestrates his bots (zombie army) using the established *C&C channel*, which needs to be robust so that it can not be easily taken down and stealthy to minimise the risk of detection. There exist two approaches for C&C channels, dictated by the flow of information between the master and its slaves, namely centralised and peer-to-peer (P2P). The two approaches are depicted in Figure 2.4.

In centralised botnets the orchestration takes place via one or more C&C servers, which are typically compromised machines themselves, to conceal the footprints of the botmaster. In early botnets such as Agobot, RBot, GTbot, DorkBot and IRCBot [22, 23, 24] the C&C, took place over an Internet Relay Chat (IRC) channel. IRC constituted an appealing choice because it allowed one-to-one and group communications, password protected channels and unlimited number of participants.

Modern botnets such as Rustock, Clickbot and Asprox [25, 26, 27] use the HTTP and the HyperText Markup Language (HTML) to issue their commands. The substitution

of IRC with the HTTP has several advantages. In particular, the prevalence of HTTP constitutes the detection of HTTP-based C&C channels very challenging. In addition, since the legitimate use of IRC has severely declined over the years, many firewall block IRC traffic as suspicious, but as far as HTTP is concerned, this is not an option.

Ultra-modern botnets such as Stegobot [28], avoid using the aforementioned protocols and instead establish covert channels by implementing steganographic techniques. Steganography relates to the practice of embedding secret information into a cover medium such as an image or an audio file; usually the recovery of the secret data requires a password. Typically, the commands are embedded in images uploaded on social media such as Facebook[15] and Twitter[16] from the botmaster. The herd periodically connects to these services to download and decode the pictures and retrieve the commands [29, 30]. Other botnets such as Koobface [31] do not use social media for C&C channels, but instead abuse them for impersonating legitimate users, setting up phishing scams and sending spam. Thanks to the popularity of social networks, network traffic that is associated with these services is often blindly considered benign and not inspected. Consecutively, the detection of botnets using this type of C&C channels is extremely onerous.

Centralised botnets have the advantage of being easy to manage and coordinate but are more likely to raise suspicions and get detected because of the unjustifiable number of machines spread all over the world connecting to the same remote host. More specifically, if the address of a C&C server becomes known, it is automatically enlisted in blacklists maintained and distributed by the scientific community and access to it can be trivially restricted using a conventional firewall.

The inherent limitations and weakness of botnets with centralised infrastructure has led cyber-criminals into decentralising their armies. Decentralised botnets such as Nugache and Storm [32, 33] are very similar to traditional peer-to-peer networks. Removing the need for a centralised server makes the botnet more robust, since there is no longer a single point of failure, and renders localisation of the commander very strenuous. In addition, blacklisting is no longer an efficient method to block access to the C&C channel, since commands may be received from thousands of different hosts. The commands are issued to one or few of the slaves, and are then distributed through the botnet. The main drawback of the decentralised approach is the additional complexity in managing the slaves and the slow reaction time, as the propagation of commands is not as fast as in the centralised modus operandi.

The two approaches of managing a botnet are not necessarily mutually exclusive, hybrid botnets like SpamThru use both types. As a result, they have the perks of both methods, achieving efficient management, flexibility and robustness [32].

[15] `https://www.facebook.com/`
[16] `https://twitter.com/`

As discussed earlier, the commander has full functional control over the infected machines and uses them for nefarious purposes. In particular, bots can be used for identity theft and financial profit. This is achieved by intercepting e-mail and social networks credentials as well as bank card numbers. Apart from the financial gains, impersonation of the victim to online platforms and services facilitates the expansion of the botnet. In addition, the zombies are often used to perform distributed denial of service (DDoS) attacks to extort money from companies that rely on their web-services. Cyber-criminals require a fee in order to cease the attack and stop interrupting the company's services [34]. Finally, bots are extensively used for unsolicited mass mailing (spam); according to a report published by Symantec in 2014 [35], approximately 76% of spam e-mails were sent by bots.

Altough IRC botnets have lost the glamour, they exist even today. Since legitimate IRC traffic is nowadays very uncommon, a host connecting to an IRC channel, would indicate that the host is part of a botnet. Following our earlier discussion, the scientific community publishes lists with IP addresses known for hosting C&C servers; detecting connections to one of these addresses would signify that the host is probably infected with a bot. In P2P botnets the victims connect simultaneously to unassociated IP addresses spread all over the world, which diverges from the typical internet user's behaviour. The detection of such connections would constitute a strong indication that a host is likely to be part of a P2P botnet.

Spammers

The term spam refers to unsolicited emails typically containing advertising content. *Spammers* are malicious programs that utilise the victim's machine to connect to SMTP servers and send bulk emails. Cyber-criminals use *spammers* to assist the propagation of their malware by attaching malicious files to their messages. In addition, spam emails are often sent as part of a phishing scam. Finally, following our earlier discussion on botnets, botmasters often exploit their servants to send thousands of unwanted emails without exposing themselves.

2.2.3 Blended Threats

We conclude this chapter, by presenting the family of *blended threats* also referred as *hybrid threats*. Although during the early decades of malware, each family was relying on solely one spreading mechanism for its propagation, this is no longer the case. Nowadays, several families rely on more than one means of propagation to increase the probability and speed of infecting new machines [36]. It is not uncommon for file infectors to exploit the internet connection of the victim to seek their new targets. Notable example is the Nimba family, which is a sophisticated virus with worm-like components.

Chapter 3

Traditional Detection Techniques

In this chapter, we discuss traditional malware detection and classification techniques and explain how they are often evaded by cyber-criminals. We believe that awareness of the different detection mechanisms and their limitations is essential to understand the necessity for more sophisticated automated detection and classification systems. The rest of this chapter is organised as follows:

- In section 3.1 we discuss signature-based detection systems.
- Section 3.2 presents heuristic-based detection methods.
- Section 3.3 is an overview of static analysis techniques.
- In section 3.4 we scrutinize dynamic analysis for malware detection.
- Finally, section 3.5 provides a high-level description of honeypots.

3.1 Signature-based Detection

Signature-based detection is the most prevalent malware detection technique, as it is deployed by the overwhelming majority of AV programs as well as intrusion detection (IDS) and prevention (IPS) systems. Signature-based approaches fall in the larger category of misuse-based detection systems, since they rely heavily upon existing knowledge of malicious files.

When a suspicious file is captured from an AV company, experts or automated systems, analyse it and monitor its behaviour. If the file is classified as malicious, a unique corresponding signature is generated and stored in the signatures database. Typically, the signature is generated using a hash function such as MD5 [37] or SHA-1 [38]. Anti-virus programs search for malicious files, by calculating the hashes over new and existing files and comparing them against the signatures database. Approaches that rely on signatures are very popular because of their relatively easy implementation and their extremely low

false-positive rates (FP), i.e. the classification of a benign file as malicious [39]. Moreover, the compact size of signatures and their consistent format, facilitates sharing lists across organisations.

The efficiency of signature-based detection has been proven to be limited. Its most critical weakness is that, similarly to all blacklist approaches, it can not prevent unseen threats. Cyber-criminals use a large vector of techniques to make their files undetectable by signature-based detection systems. At this point, we need to clarify that owing to the mathematical properties of hash functions, the slightest modification of a file, yields a completely different signature. Malware authors exploit this by encrypting, obfuscating and packing their programs.

The encryption of malicious programs protects their source code and alters the binary file. Intuitively the modified binary corresponds to a new signature that is not included in AV databases. Obfuscation techniques alter the source code of a program, while its semantics and functionality remain intact. Obfuscation of a program renders their source code difficult to read and hinders their decompilation and analysis. Similarly to encryption, obfuscation techniques modify the binary and consequently its signature. A packer is a program which compress data, hence reducing their size and altering their binary code. Packed malware are impossible to read and analysed without the corresponding unpacker. Upon execution of the packed program, the packer decompresses it and loads its code on the system's memory.

All the above techniques have been proven to be effective in evading signature-based detection [40, 41]. Depending on the number of variants and the way a malicious program creates them by employing the aforementioned techniques, malware can be classified as oligomorphic, polymorphic and metamorphic. For an in-depth analysis of the aforementioned techniques the reader is referred to [42].

3.2 Heuristic-based Detection

The inherent limitations of signature-based detection along with the exponentially growing size of signature databases, encouraged the scientific community and researchers to devise more effective detection methods. The explosion in numbers of malware using code mutation techniques, producing numerous variants of themselves, demanded for the gradual deprecation of methods relying on static signatures. Heuristic analysis is a proactive detection method and aims to detect unknown threats as well as variants of known malware. Heuristics-based approaches attempt to emulate the procedures followed by a malware analyst to understand the behaviour of malicious programs and among others involve file emulation and generic signatures.

File emulation refers to the process of emulating the execution of a program using visualisation techniques or a sandbox. Emulation techniques overcome the obstacles imposed

by encryption, obfuscation and packing. During emulation the program's behaviour is monitored and compared against behavioural patterns extracted from known malware. Depending on the number of matching criteria, a score indicating the probability that the file is malicious is assigned to it; if the score is over a predefined threshold, the file is classified as malicious. Generic signatures refer to signatures extracted from the immutable parts of a malicious program, i.e. strings or bytes that remain intact even when the program is encrypted or obfuscated. Other ways of creating generic signatures involve fuzzy hashing as well as the use of wildcards and regular expressions. The generic signature extracted from a certain file can be used to detect multiple of its variants; hence limiting the effectiveness of oligomorphic and polymorphic malware [43].

Overall, heuristic-based systems tackle the limitations of signature-based detection, as they can detect novel malware and they are not trivially evaded by self-modifying code and obfuscation techniques. This is apparent, considering that even when a malicious program is obfuscated, encrypted or partially rewritten, its behaviour stays intact. Unfortunately, the benefits of heuristics-based systems come with the cost of increased false positive rates. In particular, it is not uncommon for benign programs to behave in ways that share similarities with malicious behaviour, thus yielding false alarms.

3.3 Static Analysis

Static program analysis is the practice of understanding a program without executing it. It is typically performed as part of the code review procedure to achieve code quality and detect possible bugs. Among others it may involve data and control flow analysis as well as taint and lexical analysis. A detailed description of the above techniques is considered beyond the scope of this paper, for further information the keen reader is referred to [44].

Static analysis constitutes an attractive method for malware analysis because it does not require the existence of run-time infrastructure. Reverse-engineering a malicious program is a strong tool in comprehending its behaviour [45, 46]. Typically, it involves program dissasembly, i.e. the extraction of human readable assembly code [47], and program decompilation, i.e. the generation of higher-level semantics [48]. Based on the instructions used and the inferred high-level semantics, information about the behaviour of the program can be deduced. Security experts investigate the instructions used, and search for red flags such as suspicious strings and dynamic link libraries (dll) invoked as well as for encrypted or packed code.

The major advantage of static analysis is that execution of the sentiments is not a prerequisite, which is beneficial in terms of both cost and safety. On the downside, static analysis demands manual effort to comprehend the program's code and suffers from inherent limitations as shown in [49, 50, 51]. In particular, sophisticated obfuscation of the file and the use of complex packers can thwart the analysis. In addition, static analysis requires the binary representation of the malware, which may not be available.

We conclude that static analysis constitutes a helpful technique when it is used as a complementary method to dynamic analysis techniques described in 3.4.

3.4 Dynamic Analysis

Dynamic analysis is the practice of executing a suspicious file in a virtualised environment and monitor its behaviour to deduce whether or not it is malicious. The testing environment is typically a virtual machine or a sandbox, while bare-metal approaches have been used as well [52]. Analysing the behaviour of a program, instead of its binary code and meta-data overcomes the obstacles imposed by encryption, obfuscation and packing techniques [53]. Particularly, upon execution inside the virtual machine, the file typically decrypts, deobfuscates or unpacks itself to infect the system. While the malware is executed, analysts examine its behaviour to identify indicators of malicious activity. Among others the analysts inspect the network traffic produced, the creation of files, function calls and the memory and CPU load.

Dynamic analysis is also used for creating behavioural signatures for known malware, locating C&C servers and keeping blacklists of hosts involved in malicious activities up-to-date. Because of the diversity of malicious programs and the fact that the majority of malware are designed to infect a specific operating system (OS), the analysis is undertaken in different environments and with a large number of different inputs. This achieves the program traversing in many different execution paths and provides a complete image of its behaviour [54].

Overall dynamic analysis is a powerful method that effectively tackles obstacles faced by other detection techniques, and there is a substantial body of literature relying on techniques as such [55, 56, 57, 58]. However, it is not panacea; cyber-criminals have developed techniques to circumvent the analysis of their programs. It is not uncommon for malicious files to be capable of detecting that they are being executed in a virtual environment and adjust their behaviour accordingly [58]. Since monitoring is limited to the paths executed, malware that imitate the behaviour of benign programs can successfully evade detection. Researchers attempt to bypass this malware defence by executing the sentiments in transparent bare metal environments [52, 59].

A malware family specially designed to evade dynamic analysis is the *logic bomb*. Logic bombs are programs whose malicious payload is executed if and only if some predefined conditions are met. Logic bombs that are time-triggered are typically referred as *time bombs*. The detection of Logic bombs constitutes a very challenging task, nevertheless, the scientific community has proposed several approaches for discovering them [60, 61].

Finally, it is worth mentioning that dynamic analysis is considered particularly expensive compared to other approaches in terms of both computer and human resources. Effective analysis requires the existence of run-time infrastructure configured with multiple

environments, as well as dedicated analysts carefully inspecting the execution.

3.5 Honeypots

A honeypot is an intentionally vulnerable system for the purpose of attracting cyber-criminals. The exposed system typically appears to be part of a corporate network, while in fact it is isolated inside a demilitarized zone (DMZ). This gives the attackers the illusion of successfully breaching the security mechanisms of a real network so that they proceed with their attack. While the attackers perform their nefarious activities, all their actions are being monitored and logged from the system. The honeypot gathers information including system logs and network traffic.

Honeypots have a dual role; they act as decoy machines driving the attention of cyber-criminals away from production systems and they help gaining an insight into attack methodologies and strategies used by black hats. The attackers typically install malware on the breached machine to gain permanent access for subsequent visits. This renders honeypots an excellent way of capturing malicious programs. The captured binaries undergo careful inspection by malware analysts, who create corresponding signatures to update AV databases and secure systems against the novel threat. Insight gained into attack methodologies assists in improving IDS and IPS rules and overall keeping up-to-date security mechanisms. Moreover, because these systems are configured to have numerous vulnerabilities, they are very likely to be the first infected by a novel malware, containing this way malware outbreaks.

Similarly to all the techniques discussed in this chapter, honeypots have limitations. Cyber-criminals have found ways to discern honeypots and avoid detection and captivation of their programs [62, 63]. More specifically, because of ethical reasons, typically honeypots are configured to not participate in real attacks or spam activities, this can be leveraged from cyber-criminals to distinguish them from real systems [64].

In this chapter we discussed some of the traditional malware detection and classification techniques, a comparison between them is illustrated in Table 3.1. Although none of the techniques is silver bullet, they have the potential of achieving great results when carefully combined.

Table 3.1: Comparison table of conventional malware detection techniques.

Detection Method	Detecting unseen threats	Execution not required	Low complexity	Obfuscation resilience	Low FP rate
Signature based		✓	✓		✓
Heuristics-based	✓	⊖	⊖	✓	
Static analysis	✓	✓			
Dynamic analysis	✓			✓	✓

✓: always , ⊖: potentially

Chapter 4

Contemporary Approaches in Malware Detection

In chapter 3, we discussed some conventional malware detection and classification techniques pointing out their limitations. This chapter focuses on state-of-the-art approaches that leverage the advantages of traditional methods combined with modern technologies such as machine learning, achieving improved efficiency and resilience to malware defensive mechanisms. Earlier we put forward the claim that because of the large number of malware samples collected on a daily basis, malware analysis, detection and classification need to be automated tasks to achieve scalability. Therefore, we focus on automated techniques that rely on traffic analysis and machine-learning, as they constitute the foundations of our system.

Systems that rely on traffic analysis are generally considered more challenging to evade compared to signature-based or static-analysis based systems. More specifically, although applying obfuscation and packing techniques to a malicious binary are considered relatively trivial tasks, altering the behaviour of a malicious program is a much more challenging process. Moreover, the substantial number of different variants per malicious program, results in a large number of different binaries that exhibit identical behaviour. Systems inspecting network packets are not affected by code mutations, overcoming many powerful evasion techniques used by malware authors.

Although literature includes a large number of systems that use network traffic inspection for malware detection, most of them focus on a specific malware type. In particular, approaches like [65, 66, 67, 68] are limited to worm detection, while others such as [34, 69, 70, 71] focus on bots. Apart from the limited scope these approaches have, systems like Polygraph [68] are considered not scalable and consequently impractical. Other systems such as [80] were evaluated using a problematic data set, as the authors included files downloaded from sourceforge[1] as part of the benign programs data

[1] http://sourceforge.net/

set. The fact that sourceforge has been repeatedly accused for providing wrapped installers that apart from the desired program, install adware [82, 83], is likely to have affected their results. A scalable system relying on traffic-analysis, and transparent to the malware type is [72]. Another extremely lightweight system that is solely concerned with the order network events take place is Chatter [57].

Systems that inspect system calls and operation code (OpCodes) rather than network traffic such as [73, 74, 75, 76, 78], as well as hybrid systems that monitor both host and network behaviour like [81] are considered beyond the scope of this work and are not not discussed any further.

The demand for automated detection systems that would require minimal manual analysis stems from the exponentially increasing prevalence of malware. The utilisation of machine learning techniques is particularly popular among automated systems such as [57, 73, 74, 75, 76, 77, 78, 79, 80]. Machine learning based systems typically involve the establishment of a training set, the extraction of features, the training of a classifier and finally clustering or classifying new samples using the classifier built.

Systems that rely on traffic analysis and machine learning are not affected by code mutation techniques and are very effective in classifying malicious programs with respect to their family or behaviour. On the downside, the classifiers need constant training and careful tuning to be able to cope with the latest threats. The rest of the chapter is structured as follows:

- Section 4.1 discusses malware classification systems that rely on traffic analysis and machine learning, and underlie the foundations of this work.
- Section 4.2 presents an overview of two state-of-the-art botnet detection systems. Although these systems do not involve machine learning, the incorporated techniques can assist in devising potential features for our system.

4.1 Related Work

4.1.1 HTTP-based Malware Clustering

The system introduced by Perdisci et. al. in [72] aims to automatically produce network-level signatures by clustering together similar in terms of behaviour malicious programs. The system inspects traffic generated from the execution of HTTP-based malicious programs in a controlled environment and seeks similarities. In order to achieve efficient and scalable clustering, the authors implemented a multi-step clustering procedure. More specifically, clustering involves three sub-routines namely coarse-grained clustering, fine-grained clustering and cluster merging. Coarse-grained clustering is performed by inspecting the statistical features of HTTP traffic such as the number of requests and the

different methods used. The output of this sub-routine consists of large clusters containing samples with similar statistical traffic characteristics. Fine-grained clustering is applied next, by grouping together samples that exhibit structural similarity and belong to the same coarse-grained cluster. Since this step involves pairwise comparisons, applying it directly to the whole data-set, would render the system unscalable. The final step of the procedure is cluster merging and involves combining similar fine-grained clusters together. This step enables the extraction of a generic signature capable of detecting all the samples belonging to the merged cluster.

Regarding the clustering method, the authors report that single-linkage hierarchical clustering [84] performs better compared to other clustering techniques such as x-means [85]. The authors also denote the difficulties of evaluating a clustering system, a problem that directly concerns our system as well. The issue lies on the inconsistencies between the labels different AV vendors assign to a malicious file. The system introduced in the paper, uses labels from McAfee[2], Avira[3] and Trend Micro[4] whereas ours, utilises exclusively labels produced by ESET[5]. By retrieving labels from a sole source, we overcome the problem of potential discrepancies between labels originating from different vendors.

The system introduced has several limitations; the most severe is that it explicitly inspects malware that utilise HTTP. It should be apparent from our earlier discussion in 2.2.2, that a large number of malware families use P2P protocols instead of HTTP, these programs are considered out of the scope of this work. In addition, the specimens were executed for a limited time of 5 minutes; event-triggered malware such as logic and time bombs are unlikely to unleash their payload inside this time period and therefore can not be detected. Finally, a large percentage of modern malware use HTTPS to encrypt their traffic, hindering the HTTP content inspection, in which the system relies upon.

4.1.2 Chatter

Unlike most of the systems in literature that rely on AV labels for their ground-truth [81, 86], the authors of Chatter [57] manually vetted a large number of malware samples and created their own labels, avoiding any problems caused by potential discrepancies between labels originating from different vendors. Another innovation of Chatter is that unlike [53, 72, 81, 87], it does not require deep packet inspection, achieving this way, efficiency and resilience to the use of HTTPS.

The dynamic analysis component of the system is implemented with AUTOMAL [53], a windows-based sandboxed execution system. Malware samples executed in AUTOMAL produce the raw artifacts of the malicious program. These artifacts constitute the behavioural profile of the malware, and are subsequently used for devising features. Upon

[2] http://www.mcafee.com/us/
[3] http://www.avira.com
[4] http://www.trendmicro.co.uk/
[5] http://www.eset.co.uk/

extraction of the features, each is mapped to a character from a fixed list of alphabets. This enables the representation of a malware's behavioural profile as a text file, which consists of the aforementioned characters. The portrayal of features using a text document, allows for the usage of text-mining techniques that can unveil interesting patterns, abating the classification process.

The system proceeds with extracting the n-gram [88] features from each document; the number of n-grams determines the feature vector incorporated in the machine-learning component of the system. The system was evaluated using three different algorithms, namely: k-nearest neighbors, support vector machines and decision tree classifier. For further information on the aforementioned algorithms, the keen reader is referred to [89].

Overall, the authors present a very efficient automated system capable of classifying malware by analysing network traffic. Although Chatter is outperformed by similar systems in terms of accuracy [53, 72], it achieves a good balance between precision and operational complexity as it is capable of processing an order of magnitude more files compared to similar systems [57].

Similarly to most of the systems relying on dynamic analysis, Chatter's performance can be thwarted by malware producing a significant amount of noise, i.e. traffic irrelevant to the program's operation, used as distraction. Another issue stemming from the application of machine learning techniques, is the requirement for constantly training the classifier in order to keep the system updated.

4.2 Botnet Detection Using Traffic Analysis

As discussed in 2.2.2, bots constitute on of the most grievous malware threats since the infected machines can be remotely controlled and are typically used for heinous purposes. The remote administration takes place over the internet. In particular, a C&C channel is established between the zombies and the botmaster, allowing him to issue commands to his servants. The scientific community has suggested several approaches for botnets detection, leveraging the fact that C&C channels produce a considerate amount of network traffic.

Approaches like [24, 70, 90, 91, 92, 93, 94] are considered inadequate since they make strong assumptions about the type of the C&C channel and the structure of the botnet. In particular, [24, 90, 91, 93, 94] explicitly detect IRC botnets, while [70] focuses on botnets with centralised structure. At this point, we should clarify that the era of IRC botnets is coming to its end; cyber-cryminals prefer using HTTP or P2P protocols to harden their network's infrastructure. Therefore, although some of these systems were proved to be effective in botnet detection 5 years ago, we consider them deprecated and we emphasise that modern botnet detection systems should be independent of the communication protocol used. Two systems transparent to the botnet's structure, the

communication protocol used and without requiring specific knowledge over the payload used are BotHunter [69] and BotMiner [34].

4.2.1 BotHunter

Centralised and P2P botnets significantly differ in terms of the network traffic they produce. However, the detection of similarities between traffic produced by hosts that constitute part of the same botnet is plausible. Indeed, centralised botnets are easier to detect because a large number of hosts simultaneously receive the exact same commands. However, the fact that in P2P botnets the same commands are exchanged between neighbors, i.e. geographically close hosts, and therefore their network activity is identical can be leveraged to detect such networks. BotHunter exploits such similarities between the network traffic of different hosts to detect botnets. The system relies on the "bot infection dialog model" that models the infection life cycle as a finite set of predefined actions [69]. More specifically, the infection model includes target scanning, infection exploit, binary egg download, C&C channel establishment and outbound scanning.

Target scanning refers to the activities prior to the infection, such as port scanning to detect services running on the target machine. Infection exploit encompasses the procedure of exploiting a host vulnerability and compromising him. Binary egg download and execution describe how the malicious payload is downloaded to the victim's machine and gets executed. During the C&C channel establishment the infected machine attempts to connect to the C&C server. Finally, outbound scanning refers to the attempts undertaken by the new member of the botnet, to further spread by scanning other hosts.

BotHunter utilises the correlation engine of the popular IDS SNORT [95], along with two plug-ins tailor-made for botnet detection,namely SCADE and SLADE. SLADE performs deep-inspection of incoming packets, and takes advantages of discrepancies in byte distribution of specific protocols that are closely associated with C&C channels, performing n-gram payload analysis of network traffic. SCADE inspects port activity on both incoming and outgoing network packets and aims at detecting mainly activities that are part of either the first or the last life-cycle phase. Finally BotHunter correlates the alerts produced by Snort with information produced from its two plug-ins, to detect patterns matching its infection dialog model.

While most of botnet detection systems relying on traffic inspection require offline analysis of network packets, BotHunter can perform real-time detection. In addition, BotHunter is not limited to a mere detection of the compromised hosts, but it is also capable of inferring the source of the infection as well as the C&C server used [69].

Nevertheless, BotHunter has limitations; in particular, modern botnets encrypt their C&C channels and conceal their footprints. Since one of the main components of BotHunter, SLADE, relies on deep-packet inspection; encryption of the communications be-

tween the botmaster and its zombies is likely to have a deleterious effect on its performance. In addition, BotHunter heavily depends on its bot infection model. Although at the moment this model encompasses the typical steps of an infection, a future paradigm shift of the infection stages would require the re-configuration of the system. Finally, the system is more efficient in detecting new infections rather than existing ones. In particular, when deployed in a network with a priori compromised hosts, they system misses several stages of the infection model and thus has limited effectiveness.

4.2.2 BotMiner

Another botnet detection system independent from the botnet's structure and protocols used, but at the same overcoming the limitations imposed from relying on a predefined infection model is Botminer [34]. Botminer was designed to be capable of detecting unseen threats, therefore it does not require knowledge over botnets signatures, captured binaries or blacklists with known C&C server addresses. The system relies on the fact that compromised hosts perform similar routines and produce periodically identical traffic. In particular, bots apart from occasionally receiving commands, also exchange heartbeats, i.e. keep-alive messages, with their neighbors.

Botminer consists of 5 modules namely C-plane monitor, A-plane monitor, C-plane clustering, A-plane clustering and Cross-plane correlator. The C-plane monitor is responsible for monitoring network traffic, while the A-plane monitor detects malicious activities such port scans and spamming. The C-plane clustering processes the output of the C-plane monitor and clusters similar traffic. The A-plane clustering module receives as input the logs of A-plane monitor and groups together activities that share similar patterns. Finally, the Cross-plane correlator combines the results of the 2 clustering modules and decides whether a host is infected or not.

Botminer constitutes one of the most flexible detection systems, making minimal assumptions regarding the botnet's characteristics and achieved high precision and low FP rate in the experiments conducted [34].

However, there exist several ways in which bots can elude detection by BotMiner. Botnets introducing randomisations to the packets exchanged in the context of the bot-botmaster or bot-neighbors communications can evade clustering of the C-module and consequently detection. In addition, Botminer uses white-listing of reputable addresses such as popular social media, to achieve scalability. As a result, botnets that use the steganographic techniques discussed in 2.2.2, are undetectable from BotMiner.

In this section we discussed two state-of-the-art approaches to botnet detection that do not make assumptions on the structure of the botnet and the underlying protocols used. However, both systems require view of a large network such as a large local area network (LAN) or an ISP's network and they are not suitable for detecting compromised computers inside a network with few machines or for host-based detection.

Chapter 5

Data Collection and Initial Analysis

This chapter provides the reader a high-level description of the malicious programs our repository consists of. In particular, we discuss how these files were executed in a controlled environment to produce network traffic. We believe that awareness of the characteristics of the malware included in our data set is fundamental in grasping the capabilities and limitations of our system. In addition, we explain the procedure of retrieving the labels corresponding to our specimens. Finally, we provide some interesting findings that emerged from our analysis. The rest of this chapter is structured as follows:

- Section 5.1 describes the procedure of collecting the samples included in our corpus.
- Section 5.2 presents the limitations inherent from the nature of our data set.
- In section 5.3 we provide an overview of the approach taken in retrieving the labels corresponding to our malware samples.
- Section 5.4 portrays the geolocation of the IPs visited by our samples.
- Finally, in section 5.5 we discuss some of our most interesting findings.

5.1 Data Collection

All the files were downloaded from the Information Security Center of Georgia Institute of Technology (gtisc). The specimens were executed in the Platform for Architecture-Neutral Dynamic Analysis (PANDA[1]), which is a platform for dynamic analysis based on QEMU [96] and LLVM [97]. The record/replay logs, the packet captures (pcaps) and the corresponding MD5 digests of the specimens are stored in a publicly available repository[2] to assist the community of malware analysts. We downloaded 15,063 pcap files along with their MD5 hashes, each corresponding to a distinct malware sample.

[1] `https://github.com/moyix/panda-malrec`
[2] `http://panda.gtisc.gatech.edu/malrec/`

5.2 Inherent Limitations of the Dataset

As mentioned earlier, our data set consists of pcaps containing the traffic produced by malicious programs executed in PANDA. The specimens were executed in the Windows environment for a period of $10-15$ minutes. Hence, malware designed to infect explicitly other operating systems such as UNIX or Mac OS probably demonstrated a behavior different from the one anticipated or even aborted execution. We removed from our data set pcap files that were particularly small, to make our analysis more reliable, unaffected from this type of samples. Additionally, as a result of the rather short execution time, some programs may not have shown their full potential. In particular, it is high unlikely that any logic or time bombs have triggered their malicious payloads and consequently we do not attempt to classify any samples as such.

5.3 Retrieving Labels

Since we did not perform the dynamic analysis of the malware samples ourselves, but instead we collected the execution logs, our ground truth relies on labels provided by AV vendors. To enable the automatic retrieval of the labels corresponding to our samples, we integrated our system with VirusTotal[3] using its public API. It should be clear from our earlier discussion in 4.1.2 that because of inconsistencies in the labels different vendors provide, relying on labels from multiple vendors is problematic. Therefore, after reviewing the report of VirusTotal for a substantial number of samples, we concluded to use labels solely from ESET, owing to its significantly high detection rate and the consistency of the provided labels.

In addition, ESET recently launched a beta version of Virus Radar[4], which provides detailed descriptions of many malware families and their variants. Information provided by Virus Radar such as network protocols, ports and IP addresses used, was a great assistance during the feature extraction process and the establishment of our ground truth.

Collection of the underlying labels enabled us to group our samples and proceed with further analysis of the corpus. In particular, we inferred the spreading mechanism for each family based on information retrieved from Virus Radar and VirusTotal and we grouped our specimens accordingly.

During subsequent analysis, we noticed a significant number of variants for each malware family. For example the *Virlock* family has the variants Virlock.J, Virlock.G and Virlock.C. The part of the label that denotes the variant was considered superfluous and we chose to omit it. This was done by replacing the existing labels with their sub strings ending with the full-stop character. Removing redundant information from the labels facilitated the process of grouping together samples belonging to the same family.

[3] `https://www.virustotal.com/`
[4] `http://www.virusradar.com/`

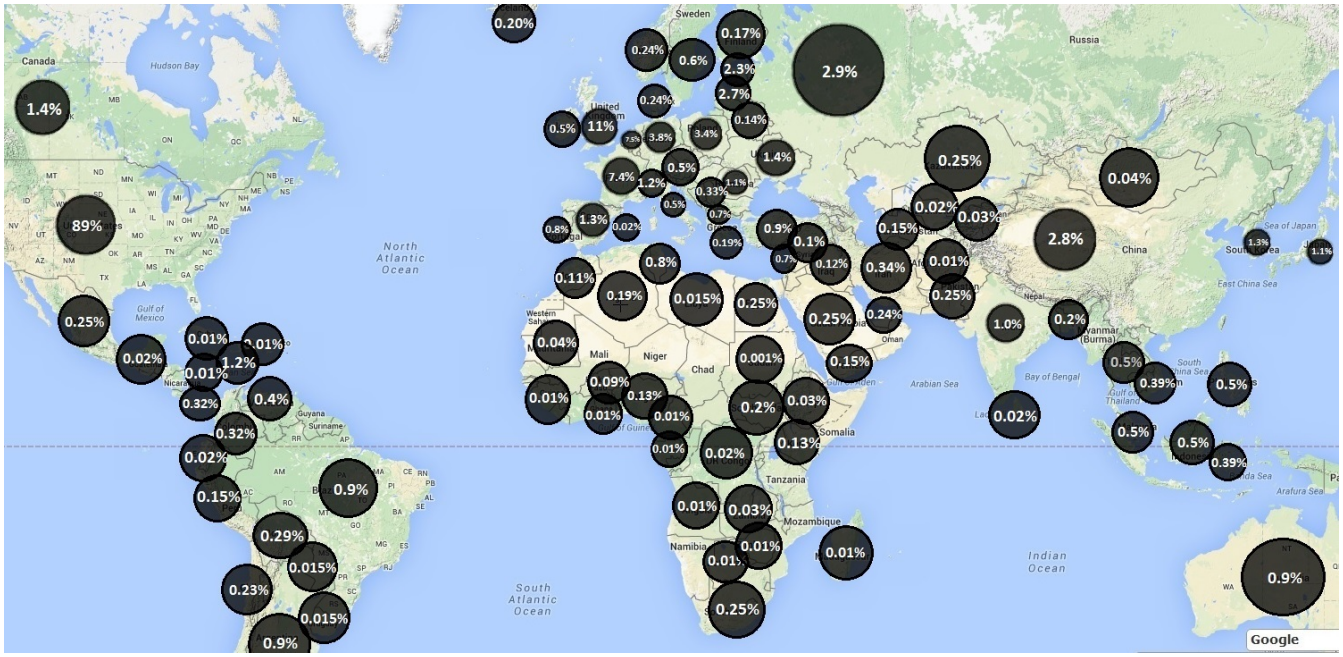

Figure 5.1: World map illustrating the percentage of malware visiting each country.

5.4 Geographical Distribution

We surveyed several aspects of our samples behaviour such as the countries visited, the DNS servers queried and the web services invoked. This type of information proved to be remarkably helpful during the process of devising features. In addition, we computed the prevalence of the visited IP addresses and countries.

We used the WHOIS protocol to resolve the domains of the visited IP addresses, and the GeoIP API to determine the countries where these IP addresses belong. Knowledge over which countries and domains each sample visited, allowed us to derive fine-grained features that rely on the higher-level semantics inferred, instead of plain traffic.

The utilisation of the GeoIP API enabled us to determine which countries each sample visited; Figure 5.1 illustrates the percentage of our samples visiting each country. We anticipated that the results would be biased in favour of the United States (US) since the samples have been executed in the state of Georgia. Nevertheless, we claim that this has not affected the results concerning the rest of the countries. The countries following the US (89%) are the United Kingdom (11%), the Netherlands (7.5%), France (7.4%) and Germany (3.8%). The most visited African country was Tunisia (0.8%), while the Latin America countries with the largest number of hits were Brazil (0.9%) and Argentina (0.9%).

5.5 Interesting Findings

We proceed with presenting some of the most striking findings that emerged from inspecting our corpus:

- Forty distinct DNS servers were queried by our samples. After manual inspection, we deduced that most of them are currently unavailable. We assume that these DNS servers are set-up by cyber-criminals, and are periodically relocated.
- A significant number of clickers and flooders visited social networks such as Facebook, Twitter and LinkedIn.
- Some other programs flooded virustotal with thousands of queries attempting to perform a DoS attack against the web site.
- Several samples queried web-services that provide information about the (external) IP address of your machine. The program can learn this way the real IP address of the infected machine, and avoid being tricked by a sandbox which provides it with a fake one.
- The worm Mydoom connected to several universities and colleges, such as the University of Chicago and Western Nevada College. Mydoom exploited the SMTP servers of these schools, to send spam and spread itself. We believe cyber-criminals target the SMTP servers of educational institutions as they are capable of handling a large amount of traffic and they have often loose security mechanisms in place.
- We discovered malware that invoked URL shortening services such as bitly[5] to create shortened links to other domains. We assume this practice serves as a mechanism to bypass any blacklists that rely on domain names.
- In addition we detected an overlap between samples visiting Twitter and bitly. We estimate that these samples invoked services as such to reduce the length of URLs, since Twitter enforces a limited number of characters per post.
- Finally, only 12 samples of our corpus connected to IRC channels, this verifies the continuous decline of botnets that use IRC for their C&C channels.

[5] `http://bitly.com`

Chapter 6

System Overview

This chapter provides a detailed description of our system. Unlike other systems included in literature, which mix the spreading mechanism used by a malicious program with its underlying behaviour and purpose, we take a different approach. In particular, we created 2 classifiers, based on the propagation method and behaviour employed by malicious programs. Our system relies on traffic analysis and machine learning techniques. Traffic analysis was performed using the JAVA API *jNetPcap*[1], a powerful API that enables dissection of network packets. For the machine-learning aspect of our system, we utilised the popular data mining and machine-learning tool *Weka*[2]. The rest of this chapter is organised as follows:

- Section 6.1 provides an overview of the first component of our system, which classifies malicious programs with respect to their propagation method.
- Section 6.2 presents the second component of our system, capable of inferring malware types with respect to the programs' behaviour and purpose.

6.1 Spreading Mechanism Classifier

This section gives an overview of our classifier operating on malware spreading mechanisms. It should be clear from our earlier discussion in 2.2.1, that depending on the propagation mechanism used, malware can be classified as viruses, worms or Trojans.

The rest of the chapter is structured as follows. Section 6.1.1 presents the procedure of establishing the ground truth of the classifier. Section 6.1.2 discuses the methodology followed in devising features describing the 3 different propagation methods. Finally, in section 6.1.3 we present and discuss the results, and analyse the factors that we believe hinder the performance of our classifier.

[1] http://jnetpcap.com/
[2] http://www.cs.waikato.ac.nz/ml/weka/

6.1.1 Ground Truth Establishment

In this section we focus on describing the procedure of establishing the ground truth on which the spreading mechanism classifier relies upon. As discussed earlier, we retrieved the labels corresponding to our samples from ESET by querying Virus Total. ESET was preferred over other vendors because of its Virus Radar platform, which provides detailed descriptions about each malware family and its variants. Using information from Virus Radar, we inferred the spreading mechanism of $7,201$ malware samples. We proceeded with splitting our data set in 3 sub-sets, one for each spreading mechanism, creating the following groups:

Table 6.1: Samples distribution among the different spreading mechanism types.

Class	**Trojan**	**Worm**	**Virus**
Samples	4618	651	1932

6.1.2 Devising Features

For each sub-group, we computed a large vector of statistical metrics including the volume of the traffic produced, the prevalence of different protocols and the number of ports used. For each metric we calculated the mean as well as the median values. Although the mean is the most common metric, it is strongly affected by outliers and thus is considered unreliable by itself. On the other hand the median is resilient to extreme values and therefore is regarded as more reliable. Knowledge of both the mean and median values for each metric allowed us to infer the coherency of its each sub set and the representativeness of the features. Table 6.2 illustrates the results for some of the statistics we computed.

Table 6.2: Traffic Analysis Statistics.

	Trojan		**Worm**		**Virus**	
Metric	**Mean**	**Median**	**Mean**	**Median**	**Mean**	**Median**
Packets	$5,546$	873	$7,563$	511	284	301
Packets sent	$1,385$	341	$3,704$	207	134	144
Packets received	$4,084$	500	$3,795$	250	91	105
Bytes sent	109.4 KB	27.3 KB	385 KB	11.9 KB	8 KB	10 KB
Bytes received	5.3 MB	480 KB	650 KB	230.4 KB	47.8 KB	61.2 KB
Source ports	108	45	133	42	33	35
Destination ports	91	39	123	32	27	29
IP Packets	$5,518$	817	$7,501$	458	226	250
TCP Packets	$5,276$	746	$7,257$	370	160	184
UDP Packets	188	72	231	98	60	60
HTTP Packets	273	56	187	16	25	32
DNS Queries	43	15	52	11	8	9

Other statistics, not depicted in the table, included the utilisation of different protocols such as FTP and ICMP, the number of distinct DNS servers queried, the distribution among the different TCP flags, the prevalence of specific ports and the number of established TCP connections.

The calculation of these statistics allowed us to detect notable differences between the different malware types. In particular, all three categories received more bytes than they sent. In addition, the number of bytes Trojans received was substantially larger compared to the others. This was anticipated, as our set includes a significant number of dropper samples, which are expected to download large files from remote hosts. By examining transport layer traffic, we observed that Trojans used the largest number of distinct ports. As anticipated, the IP and TCP protocols were extensively used in all three groups, while the usage of UDP was limited. Finally, by inspecting application layer packets, we deduced that Trojans involved the highest HTTP and DNS activity. We proceed with analysing the features used by our classifier.

Incoming and Outgoing Traffic

As discussed earlier, the three types exhibited different behaviours in terms of the produced network traffic, as well as the number of the bytes received and sent. The first three of our features aim to capture potential discrepancies between the three types in terms of the volume of traffic they produce. Figure 6.1 illustrates the cumulative distribution functions of the sent, received and total number of packets for each category.

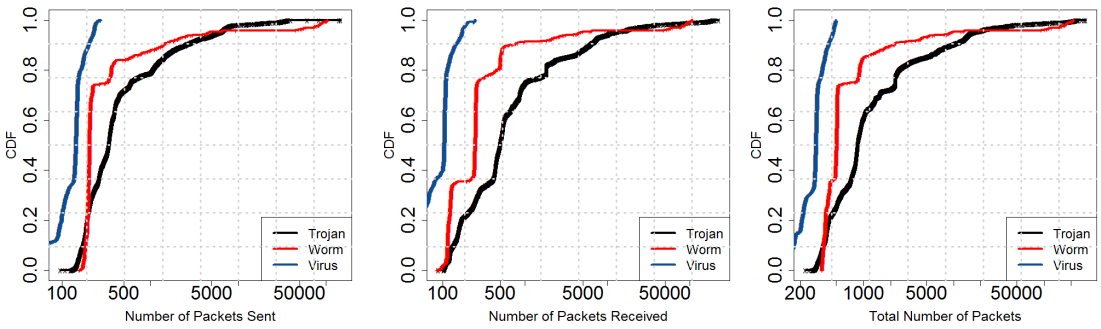

Figure 6.1: CDF plots of the sent, received and total number of packets.

The results indicate that Trojans exhibited higher network activity compared to viruses and worms. While this validated our estimations about viruses, it contradicted our expectations about worms. In particular, worms were expected to generate a tremendous load of traffic, but this was not confirmed by our analysis. Nevertheless, it is clear that the volume of traffic produced by worms is significantly larger compared to the traffic generated by viruses. By examining the similarity between the three plots, we can also infer that Trojans and worms received more packets than they sent, while that was not

true for viruses. Finally, the curves corresponding to the total number of packets do not have significant overlaps, which implies that the volume of traffic produced from each sample constitutes a robust feature.

Sent and Received Bytes

The following features are drawn from the fact that many malware samples are expected to either download or upload files to remote hosts. More specifically, we estimated that a large percentage of our corpus downloads new malicious files to the victim's system. In addition, it is common for malicious programs to update themselves to their newest version as soon as they are executed. While observing outgoing traffic, we noticed that certain families uploaded encrypted packets to remote hosts that potentially included personal information of the victims. Other samples have been found to upload a huge number of files to certain known web services in an attempt to perform a DoS attack. The plots in Figure 6.2 depicts the CDF plots of the bytes sent, received as well as the total number of both incoming and outgoing bytes.

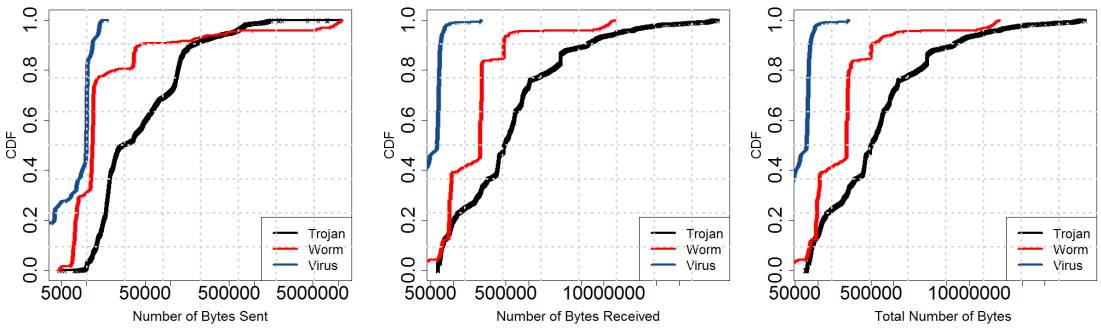

Figure 6.2: CDF plots of the sent, received and total number of bytes.

As expected, Trojans received a much higher number of bytes compared to worms and viruses. This was expected since our Trojan sub-set includes a significant number of droppers and adware samples, which have been found to receive large files from remote hosts. Viruses did not send or receive a considerable number of bytes, this corroborated our earlier discussion in 2.2.1 and confirmed our estimations about viruses being the most "stealthy" category in terms of network activity.

We expected worms to receive a smaller number of bytes, since as discussed in 2.2.1, worms are characterised by their aggressive attempts to propagate using the network, while receiving content from remote hosts is relatively uncommon. Nevertheless, the results of our initial analysis depicted in Table 6.2, indicate that worms produced a substantial volume of TCP traffic, which clearly had a beneficial effect on the number of bytes they received. Overall, we do not detect significant overlaps in any of the plots above, which indicates the robustness of these features.

IP Addresses and Countries Visited

Our initial hypothesis involved worms visiting more IP addresses and countries compared to viruses and Trojans. This is because worms typically connect to multiple remote hosts, either retrieving addresses from an embedded list or by generating random addresses in an ad hoc fashion. In the latter case, it would not be surprising if many of these addresses were located in different countries. As discussed earlier, the countries corresponding to the IP addresses visited were retrieved using the GeoIP API. Figure 6.3 portrays the CDFs plots of the distinct IP addresses and countries our samples visited.

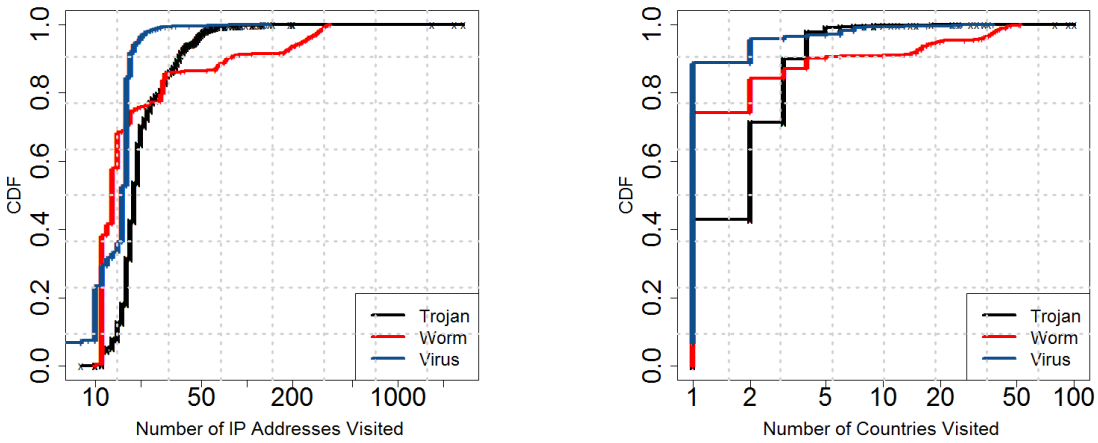

Figure 6.3: CDF plots of the IP addresses and countries visited.

While the bottom half of the first plot does not provide any particularly useful information, we observe the three curves deviating in the upper half. In this case, worms exhibited a behaviour partially consistent to our hypothesis, as about 20% of them attempted connections with more than 30 addresses, while some samples reached the massive number of 300 IPs. Viruses validated our estimations visiting at most 15 addresses, while Trojans were found to be more active, reaching their peak at 60.

Naturally, the plot illustrating the number of distinct countries visited, follows a pattern, similar to that of the first plot. The overwhelming majority of viruses visited less than 3 countries, while Trojan samples visited at most 4 countries. Finally, although a significant percentage of worms visited 3 countries at most, we discern that some samples exceeded our expectations visiting the immense number of 40 countries.

TCP Packets and Established Connections

Transport layer protocols were of particular interest; however, after careful inspection we concluded that UDP traffic did not constitute a feature helpful to our classifier. Our TCP analysis involved measuring the distributions among the different TCP flags but

the results implied that we could not use them as features as they were found to be similar. However, we observed some non-negligible differences between the numbers of (TCP) connections each type established. We considered a connection as established, if a successful 3-way TCP handshake was present. Our results on the volume of TCP packets and the number of established connections are summarised in Figure 6.4.

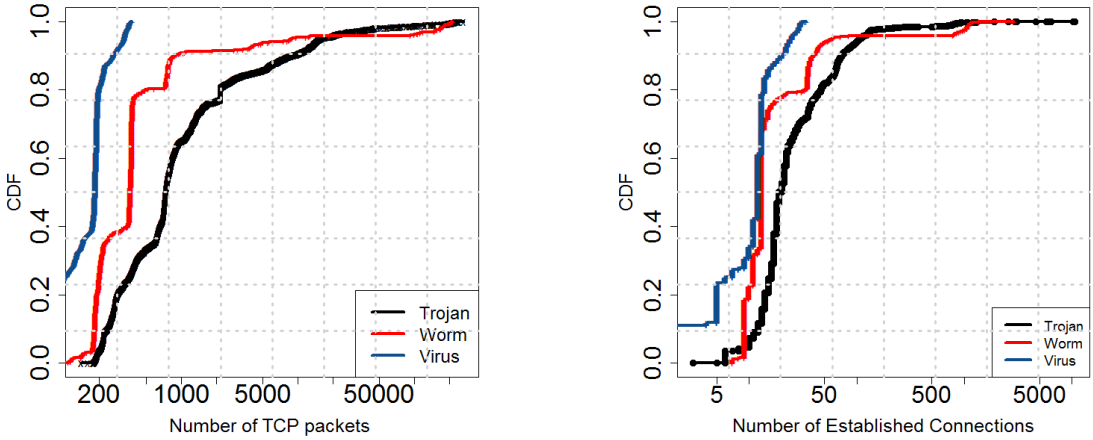

Figure 6.4: CDF plots of TCP packets and established connections.

As anticipated the CDF plot of the TCP traffic resembles the third plot of Figure 6.1 illustrating the total number of packets. It is obvious that Trojans produced more TCP traffic compared to viruses and worms. Initially we estimated that worms would involve the highest TCP activity, but clearly this is not verified by our results.

Although the two plots above share a similar pattern, we observe the three curves being closer to each other in the second plot. This implies that the samples used each connection to send a large volume of packets. The above is also verified when considering the significant overlaps between the viruses and worms curves, despite their notable discrepancies in terms of the number of TCP packets produced. Even the curve corresponding to Trojans, which unarguably produced the largest amount of TCP traffic, is not very distinct from the others. Nevertheless, during the evaluation of the classifier we found the number of established connections to be a functional feature.

HTTP

We computed statistics for several application layer protocols such as DNS, FTP and NTP but only HTTP was found to be of particular interest. We inspected different features of HTTP traffic such as the number of GET and POST requests, the hostname, the referer and the user-agent. Our findings suggested that the number of GET and POST requests, in combination with the presence or absence of the referer field,

could be leveraged to improve the accuracy of our system. Figure 6.7 summarises ours findings over the aforementioned measurements.

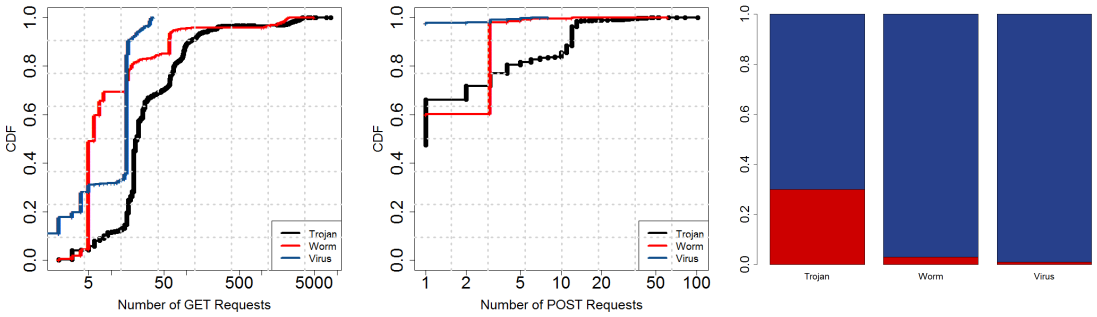

Figure 6.5: CDF plots of GET (left) and POST (middle) requests, bar-plot depicting the usage of the referer field (right).

Surprisingly enough, we observe that a non-negligible 65% of viruses sent more GET requests than worms. However, the number of GET requests sent by worms spans a large vector of values, as approximately 20% of them exceeded the number of 30 GET requests, which was the peak for viruses. Trojans dominated the other types with 40% of them sending more than 50 requests. Finally we note an overlap between the worms and Trojans curves for a mere 8% of their samples.

By examining the second plot depicting the CDF plot for the number of POST requests, it is clear that viruses sent a negligible number of them. In addition, a limited number of POST requests was sent by worms, while it is noteworthy that multiple samples sent precisely 3 requests. Trojans exhibited a more diverse behaviour, as the corresponding number of POST sent, ranges from 0 to 20 requests.

The third diagram of Figure 6.7 is a bar-plot portraying the percentage of each type using the referer field of the HTTP. Approximately 30% of Trojan samples utilised the referer field, while the overwhelming majority of HTTP packets generated by worms and viruses did not include a referer. While 30% is not considered a substantially large percentage, since the referer is absent for the majority of samples included in the worms and viruses sub sets, we estimate that this constitutes a robust feature.

Table 6.3 provides an overview of the features incorporated in our spreading mechanism classifier along with their mean and median values, as well as their χ^2 rankings.

Table 6.3: Features Overview.

Feature	Mean			Median			χ^2
	Trojan	**Virus**	**Worm**	**Trojan**	**Virus**	**Worm**	**Rank**
Total Traffic	5,546	7,563	284	873	511	301	7
Outgoing Traffic	1,385	3,704	134	341	207	144	5
Incoming Traffic	4,084	3,795	91	500	250	105	8
Bytes Sent	109,473	385,264	8,003	27,298	11,931	10,039	11
Bytes Received	5,347,509	650,074	47,840	480,042	230,398	61,236	2
TCP Traffic	5,276	7,257	160	746	370	184	1
Established Connections	54	61	11	19	13	12	4
IP addresses Visited	23.31	37.84	14.35	18	13	15	12
Countries Visited	2.09	3.76	1.32	2	1	1	6
Gets	3.03	92.2	12.6	22	6	16	9
Posts	3.03	1.35	0.09	1	0	0	10
Referer (%)	70%	0,1%	0.05%	-	-	-	3

6.1.3 Results and Discussion

As discussed earlier in this chapter, we take advantage of Weka's powerful features to evaluate our system. We evaluated our system using different combinations of classifiers and features evaluators. The Random Forest classifier yielded the optimal combination of TP and FP rates. Our results presented in Table 6.4 were computed using the Random Forest Classifier with 10-fold cross validation.

Table 6.4: Detailed accuracy by class.

TP Rate	**FP Rate**	**Precision**	**Recall**	**F-Measure**	**ROC Area**	**Class**
0.957	0.029	0.924	0.957	0.94	0.995	Virus
0.862	0.012	0.875	0.862	0.868	0.992	Worm
0.961	0.048	0.973	0.961	0.967	0.992	Trojan
0.951	0.04	0.951	0.951	0.951	0.993	

Our system achieved an overall TP rate of 95.1% and a FP rate of 4%. In particular, our system classified viruses with 95.7% TP and 2.9% FP rates, worms with 86.2% TP and 1.2% FP and Trojans with 96.1% TP and 4.8% FP rates. Although worms yielded the lowest TP rate, they also achieved the lowest FP rate. The confusion matrix of the 3 spreading mechanism categories is depicted in Table 6.5. Overall, the overwhelming majority of our samples were classified correctly, while only a small percentage of our specimens were misclassified. More specifically, 95.6% of our viruses were classified correctly, while 1.3% of them were classified as worms and 3% as Trojans. We estimate that the misclassified viruses are likely to render the victim part of a botnet, thus producing increased traffic and confusing the classifier.

Table 6.5: Confusion Matrix.

Virus	**Worm**	**Trojan**	
1848	26	58	**Virus**
24	561	66	**Worm**
128	54	4436	**Trojan**

In addition, 86.1% of worms were mapped to the correct category, while 3.6% of them were misclassified as viruses and 11.7% as Trojans. We believe that the relatively low accuracy achieved in worm classification stems from the limited number of samples belonging in the underlying category. Moreover, after manually analysing some of the misclassified samples, we discerned that their network behaviour was indeed similar to that of samples included in the Trojan and virus sub sets. We suspect that these misclassifications might be the result of ESET assigning wrong labels to some of our samples.

Finally, our system classified correctly 96.0% of Trojan samples, while 2.7% and 1.1% of them, were classified as viruses and worms respectively. We believe that Trojans which were classified as viruses might not have unleashed their payload during the analysis, for example an adware toolbar is not expected to produce any traffic since the dynamic analysis did not involve the execution of a web browser. Additionally, some Trojan families such as coin miners, are not anticipated to produce substantial traffic.

6.2 Behaviour-based Classifier

The second classifier of our system attempts to infer the type of malicious programs based on their behaviour and purpose, i.e. what is the malware sample used for. We estimate that this constitutes a more challenging task compared to our first classifier, since the number of the different categories is significantly larger and the cardinality of each sub set is significantly smaller.

6.2.1 Ground Truth Establishment

In this case the establishment of a ground truth was more straightforward, as several of the retrieved labels were self-explanatory; for example the label AdWare/MultiPlug intuitively corresponds to samples falling in the adware category. However, there existed labels that did not provide any information about the behaviour of the samples and for which we inferred their behaviour from Virus Radar. Unlike information about the spreading mechanism, which can nearly always be found online, behaviour-based information is scarce. As a result, there were several specimens for which we could not establish a ground truth because of the limited information available. Thus, we created a new data set that solely consisted of families, for which we were able to retrieve information describing their behaviour. This resulted to a new data-set including $3,943$ malicious programs with 8 different behaviours. Next, we divided our data-set into

sub-sets according to the different behaviours described in 2.2.2, creating the following sub-groups.

Table 6.6: Samples distribution among the different behaviour types.

Class	Adware	P2P Bot	Clicker	Dropper	Flooder	Coin Miner	Ransomware	Spammer
Samples	2049	142	57	935	64	62	574	60

6.2.2 Devising Features

In this section the discussion points to the procedure behind devising the features incorporated in the behaviour-based classifier of our system. Similarly to before, we computed the mean and median measures of several statistical metrics. Part of the computed results, are summarised in Tables 6.7 and 6.8.

Table 6.7: Traffic Analysis Statistics for adware, P2P bots, clickers and droppers.

Metric	Adware		P2P Bot		Clicker		Dropper	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Packets	7,843	836	1844	512	1,067	1,060	4,032	767
Packets sent	1,519	256	555	265	514	506	1,545	312
Packets rec.	6,231	518	1,231	154	483	478	2,411	383
Bytes sent	77.7 KB	19 KB	23.3 KB	11 KB	295 KB	275 KB	129 KB	18 KB
Bytes rec.	9 MB	629 KB	1.59 MB	66 KB	248 KB	231.4 KB	2 MB	164 KB
Src. ports	86	44	165	88	54	54	171	58
Des. ports	61	38	138	73	38	41	155	52
IP	7,751	787	1,787	457	998	994	3,958	721
TCP	7,554	714	1,546	232	876	868	3,645	547
UDP	191	71	234	125	115	106	306	76
HTTP	86	42	47	36	92	97	753	48
DNS	55	30	61	46	25	24	184	32

Table 6.8: Traffic Analysis Statistics for flooders, miners, ransomware and spammers.

Metric	Flooder		Coin Miner		Ransomware		Spammer	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median
Packets	41,111	52,893	362	394	272	219	89,941	93,564
Packets sent	20,681	26,447	109	81	135	107	43,568	45,071
Packets rec.	20,376	26,406	77	41	89	46	46,296	48,417
Bytes sent	20.7 MB	25.8 MB	5 KB	4 KB	7.3 KB	4 KB	6.5 MB	5.4 MB
Bytes rec.	1.7 MB	2.2 MB	43.4 KB	4.9 KB	41.4 KB	16.5 KB	7.5 MB	7.3 MB
Src. ports	43	40	26	23	33	29	1,038	1,545
Des. ports	34	33	21	18	27	21	1,025	1,521
IP	41,058	52,854	188	123	225	169	89,865	93,489
TCP	40,973	52,799	110	74	164	84	88,937	92,228
UDP	78	64	68	73	54	63	921	342
HTTP	7,526	10,562	14	4	26	10	1,691	120
DNS	22	22	11	10	14	12	780	272

We stress that some of the features described in 6.1, were found to be suitable for this classifier as well. However, during our analysis, it became apparent that new features capable of encompassing the different malware behaviours had to be inferred.

It is obvious that the spammers produced the largest amount of network traffic with a median value of 93,564 packets, followed by flooders with 52,893 packets. Furthermore, ransomware and coin miners generated the least traffic, producing 219 and 394 packets respectively. In addition, flooders dominated the other types in terms of the volume of bytes sent, sending over 25 MB, while spammers were first with respect to the number of bytes received with a median value of over 7 MB. Interestingly, the number of distinct ports used by flooders was quite small considering the amount of traffic they produced. Finally, spammers produced the remarkably high number of 270 DNS packets, while no other category exceeded 50 DNS packets.

Incoming and Outgoing Traffic

Our experience from implementing the first classifier suggested that the volume of traffic sent and received constitute robust features. This was verified by our initial analysis, as we observed significant differences in the volume of traffic produced by the different categories.

Initially, we expected spammers and flooders generating a significant load of traffic. In particular, spammers were estimated to send a large number of unsolicited mails, while flooders to launch DoS attacks against web pages. Considering the prevalence of TCP and the limited utilisation of UDP, we anticipated that a large number of sent packets would be incidental to a large number of received packets. Ransomware were expected to generate a limited number of packets, since as discussed in 2.2.2, their most characteristic feature is the encryption of the victim's file-system, which does not involve significant network traffic. Similarly, coin miners utilise resources of the compromised machine to mine Bitcoins and are not regarded as particularly network-active. Figure 6.6 illustrates the CDFs of the number of packets sent, received as well as their totals.

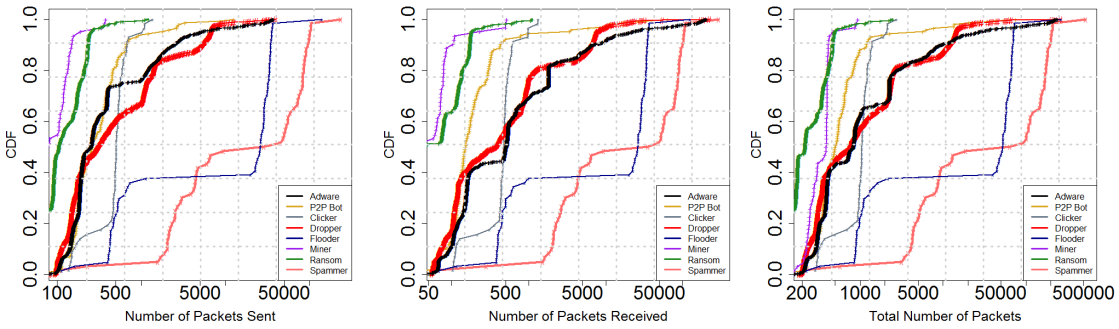

Figure 6.6: CDF plots of sent, received and total number of packets.

As expected, spammers and flooders dominate the others in terms of both incoming and outgoing traffic. In addition, we observe that ransomware and coin miners did not generate a substantial number of packets. We also discern some overlaps between adware, droppers and bots in the bottom half of all three figures, however their corresponding curves diverge in the upper half. Another interesting observation refers to the CDF of clickers, as it remains stable around a fixed value in all three plots. We believe this partially stems from the limited number of samples included in the clickers subset.

Sent and Received Bytes

Similarly to the spreading mechanism types, the different behaviour-based categories differ in terms of the number of bytes sent and received. This was also verified by our initial analysis and the statistics presented in Tables 6.7 and 6.8. Initially, we estimated that spammers and droppers would send a great number of bytes, while we expected droppers to be first with respect to the bytes received. In addition, we expected ransomware and coin miners to exchange a limited number of bytes with remote hosts.

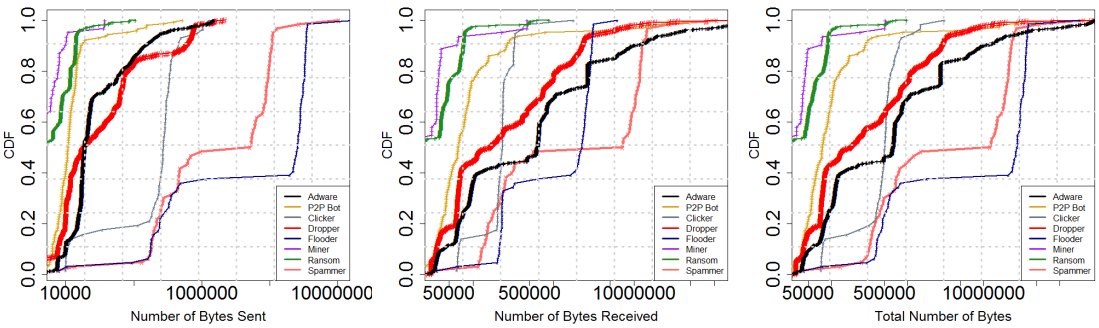

Figure 6.7: CDF plots of sent, received and total number of bytes.

The results verified our hypothesis, as clearly flooders sent the largest number of bytes, followed by spammers, with the majority of both families exceeding 10 MB. We also observe a relatively high number of bytes sent from clickers, despite the limited volume of traffic they produced. Ransomware, coin miners and bots did not sent a significant amount of bytes, as the majority of the samples belonging to these families sent less than 20 KB.

By inspecting the second plot, we discern that spammers received the largest volume of bytes compared to the other families with half of the samples receiving at least 7 MB, while some samples attained the remarkable number of 20 MB. Flooders came second with the majority of samples ranging from 1 to 3 MB. Therefore, our estimations about droppers receiving a massive number of bytes were proved false. The volume of bytes received by adware and droppers spanned a large range of values with some samples receiving less than 600 KB, while the most active ones surpassed the 3 MB. Another

interesting observation is that approximately 10% of adware samples received more bytes than spammers. Finally, although clickers sent more bytes than adware and droppers, we observe that their corresponding number of received bytes is substantially smaller.

IP Addresses and Countries Visited

Our initial analysis indicated that the number of IP addresses and distinct countries visited by each sample constituted potential features for our classifier. Our hypothesis involved the majority of our P2P bot samples connecting to numerous hosts, which are members of the same zombie army. In addition, we expected spammers attempting connections to numerous hosts, as they typically connect to multiple mail servers to send unsolicited mail. Figure 6.8 depicts the CDF plots of the number of IP addresses and countries visited by our corpus.

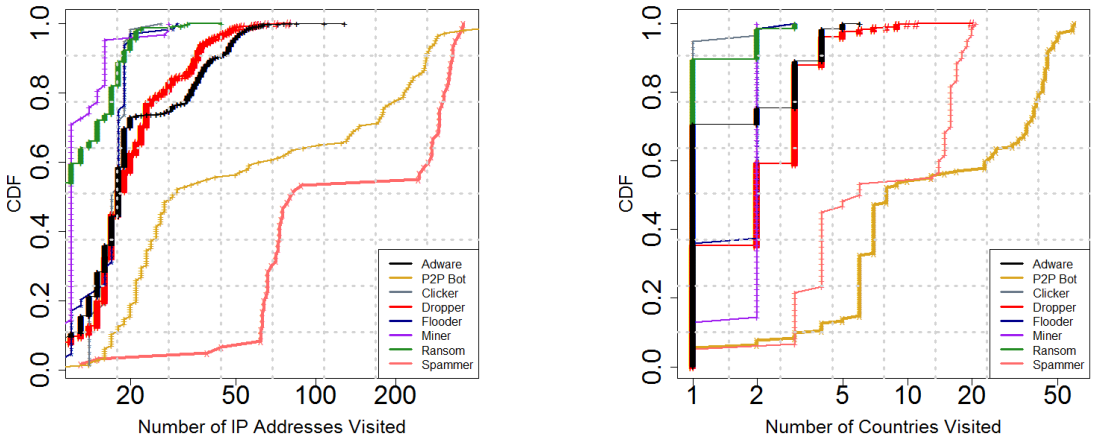

Figure 6.8: CDF plots of IP addresses and countries visited.

Our initial hypothesis was proven to be solid, as the overwhelming majority of P2P bots and spammers visited over 40 IP addresses, which was the maximum number for adware and droppers, while the rest of the families visited 20 addresses at maximum.

Although spammers attempted connections with more hosts than P2P bots, the second plot makes clear that bots connected to more countries, with 40% of the samples visiting over 20 distinct countries. This implies that spammers connected to multiple SMTP servers located in the same country, while the peers with which bots connected were scattered all over the world. Finally, the majority of the samples included in the other families visited a limited number of countries with a maximum value of 3.

TCP Packets and Established Connections

The results of our initial analysis, summarised in Tables 6.7 and 6.8, indicated the existence of significant differences between our sub-sets in terms of the TCP traffic they produced. Regarding the other transport layer protocol, we concluded that the volume of UDP traffic did not constitute a feature useful to our classifier. Taking into account the limited use of UDP, we expected that the CDF plots of TCP packets and total number of packets would be similar. Our results are summarised in Figure 6.9, the first plot portrays the CDF of the TCP packets sent and received by our samples, while the second illustrates the CDF of the established TCP connections.

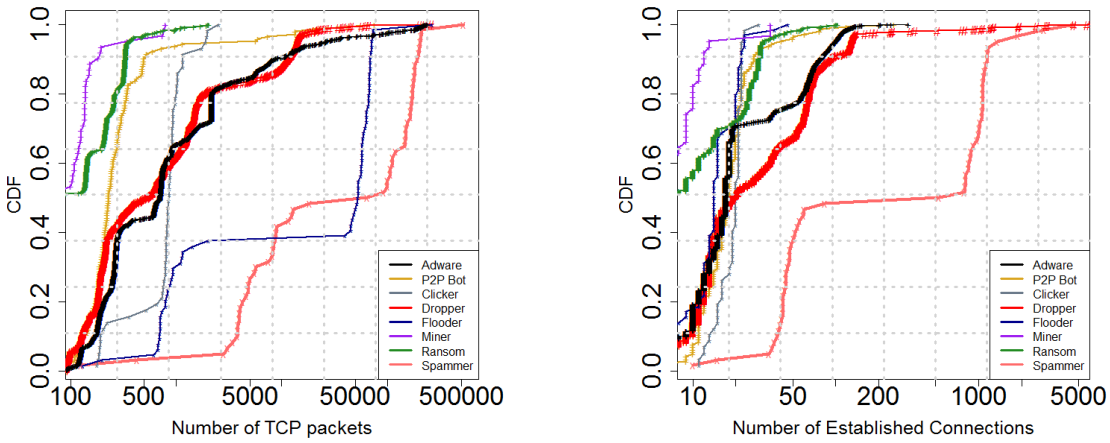

Figure 6.9: CDF plots of TCP packets and established connections.

As expected, the first plot shares many similarities with the CDF plot of the total traffic produced. Spammers utilised heavily the TCP, as they connected to SMTP servers to send a massive number of unwanted emails. Likewise, a large percentage of flooders have been found to upload a substantial number of files to web-services, launching DoS attacks. Interestingly, although coin miners produced more network traffic than ransomware overall, we observe that ransomware produced a higher volume of TCP traffic.

Although there exist some strong similarities between the CDF plots of TCP packets and established connections, we discern some worth mentioning discrepancies. Interestingly, flooders established a small number of connections considering the amount of TCP traffic they generated, with a maximum value of 25 connections. In addition, we observe that bots, clickers, adware, droppers and spammers established more connections than flooders. This indicates that flooders utilised each connection they established to exchange a immense amount of data. Despite the fact that adware and droppers generated similar numbers of TCP packets, we notice that their corresponding curves of established connections diverge for 30% of the samples of each category. Finally, spammers dominate the other families, with 50% of their samples establishing more than 100 connections,

which is the maximum value attained by both droppers and adware.

HTTP

Our initial analysis indicated that flooders and clickers sent a substantially large number of HTTP packets. As discussed earlier, flooders send a massive number of packets in their attempt to hinder the performance of specific web-services. Clickers aim to artificially increase the visitors of certain web-sites or to click on online advertisements hosted in web pages, creating revenue for the owner of the page. In addition, we estimated that the utilisation of HTTP by the other families would be limited. Figure 6.10 accumulates our findings over the produced HTTP traffic as well as the prevalence of each HTTP request method.

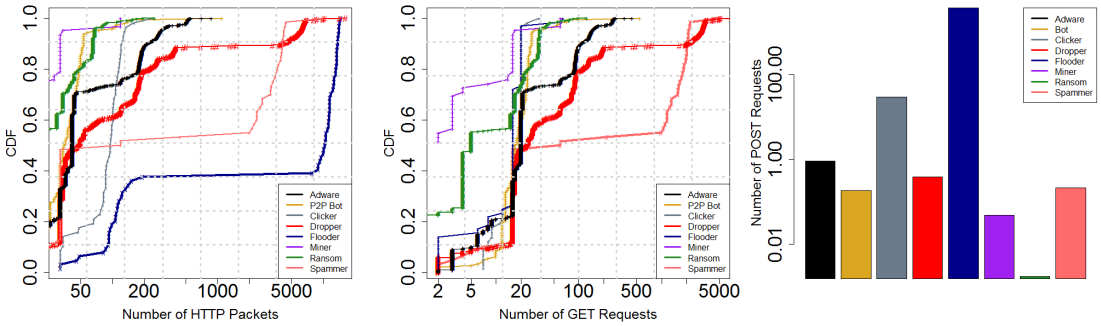

Figure 6.10: CDF plots of HTTP packets (left) and GET requests (middle), bar-plot depicting the number of POST requests (right).

Our hypothesis was proved partially correct, as flooders were found to create the largest amount of HTTP traffic. However, the number of HTTP packets sent by clickers was not as high as expected, with 80% of the samples creating between 100 and 150 HTTP packets. Although clickers did not fully meet our expectations, they still dominate all the other families in the bottom half of the plot except from the flooders, whereas the upper half of the plot makes clear that adware, spammers and droppers produced more HTTP traffic than clickers.

Our consequential analysis of the HTTP protocol, includes the prevalence of each request method, namely the number of GET and POST requests. Surprisingly, we observe that although flooders generated the most HTTP traffic, the number of GET requests they sent was limited. In particular, it is clear that the majority of flooders sent 20 GET requests, while half of the samples included in the spammers, droppers and adware sub sets exceeded that number. The increased HTTP traffic generated by flooders along with their small number of GET requests indicates that flooders primarily utilised the POST request method.

The claim above is verified from the bar plot portraying the average number of POST requests sent by each family. Clearly, flooders sent a considerably large number of POSTs with a mean value of $3,500$ requests, followed by the clickers with a mean value of 80 requests per sample. The rest of the families did not send more than one requests, while the number of POST requests sent by ransomware is negligible.

Specific Domains and Email Activity

At this point, the reader has probably realised that most of the features discussed so far, were also used in our first classifier. Indeed, our analysis indicated that some of the features used in our spreading mechanism classifier can be leveraged to infer the behaviour of different families. However, subsequent analysis denoted that in order to distinguish different, in terms of behaviour, families that produce similar patterns of network traffic, we needed to infer some new features, unique to each family. After thorough inspection, we distinguished certain distinctive features of some families.

In particular, we noticed that the majority of adware contacted advertising servers such as polmontventures[3,4], qadservice[5], adcash[6] and AdAdvisor[7]. As discussed in 2.2.2 this behaviour was anticipated from adware, as their aim is to present unsolicited advertisements to the victim, creating revenue for the cyber-criminals.

As discussed earlier in 2.2.2, depending on the structure of the botnet, the infected hosts receive their commands either from C&C servers (centralised) or peer bots (P2P). Although the samples included in our data set constitute P2P bots, and therefore they were expected to receive commands from fellow bots rather than a server, we estimated that a non-negligible proportion of them would still connect to domains known for hosting C&C servers. To detect such domains, we inspected the communications of bot samples and created a list containing the domains, which were visited from at least two of them. We proceeded with evaluating our list using information retrieved from AV vendors and publicly available blacklists such as ZeuS Tracker[8]. Our estimations were proved correct, as we discovered that a significant percentage of our bot samples connected to domains associated with malicious activities.

Typically, these domains hosted a minimal e-shop, advertising some random products to avoid suspicion, while we detected that bots connected to them by providing several parameters. A representative example of these domains housing a flower eshop, can be found in *flora-group.com.tn*[9]. Regarding domains that seemed legitimate, we suspected

[3] ads.polmontventures.com
[4] polmontventures.adk2.com
[5] ads.qadservice.com
[6] www.adcash.com
[7] adadvisor.net
[8] zeustracker.abuse.ch/blocklist.php
[9] http://www.flora-group.com.tn/

that they have been compromised by cyber-criminals and were used to assist their communications with their zombie armies. In addition, we discerned that many of these domains were in gibberish as their name did not consist of real words, for example the domain dadafarada.com. Additionally, while the most common length of a domain name is 8 characters [98], we observed that many of these malicious domains were unusually long, for example the domains liderancaspoliticas and vadivudaiammantransports consisted of 19 and 25 characters respectively.

Our final feature aims to capture the behaviour of spammers. Intuitively, spammers connect to SMTP servers and send thousands of mails. In order to detect spamming activity, we looked for potential utilisation of ports associated with SMTP and SMTP-S such as 25, 2525, 587 and 465. In addition, we inspected the messages exchanged between the malware samples and SMTP servers to determine the number of mails sent.

Our findings are summarised in Figure 6.11, which includes the bar plots portraying the percentage of samples of each category establishing connections to advertising servers, domains known for hosting C&C servers as well as the proportion of samples sending emails.

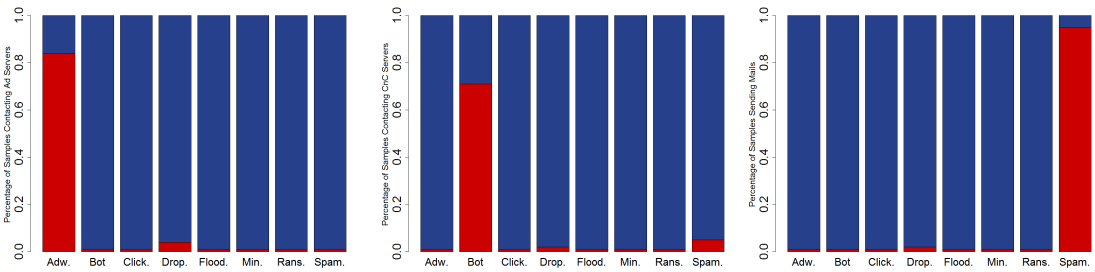

Figure 6.11: Bar plots portraying the percentage of samples connecting to advertising servers (left), domains hosting C&C servers (middle) and sending mails (right).

The above features were proved to be remarkably useful during the evaluation of our classifier. It is apparent that, there exist significant differences between the 8 categories in all the above 3 plots. In particular, more than 80% of the samples included in the adware category connected to advertising servers. Clearly the rest of the families did not connect to advertising servers, with only a insignificant 5% of droppers utilising services as such, while the corresponding percentages of the remaining families are negligible.

Likewise, the second bar plot made clear that a significant 65% of P2P bot samples connected to domains known for housing C&C servers, while the number of connections to these domains from the rest of the families were found to be very limited. Finally, spammers verified our estimations, as more than 90% of them sent at least one e-mails, while clearly the other families did not involve significant email activity.

An overview of our features can be found in Tables 6.9 and 6.10. The features related to specific domains and email activity have binary values and therefore are not included the tables below; nevertheless, the reader can infer their values from the corresponding bar plots included in Figure 6.11.

Table 6.9: Mean values of the features used.

Feature	**Mean**							
	Adware	**P2P Bot**	**Clicker**	**Dropper**	**Flooder**	**Miner**	**Ransom**	**Spammer**
Total Traffic	7,843	1,844	1,067	4,032	41,111	362	272	89,941
Outgoing Traffic	1,519	555	514	1,545	20,681	109	135	43,568
Incoming Traffic	6,231	1,231	483	2,411	20,376	77	89	46,296
Bytes Sent	77.7 KB	23 KB	295 KB	129 KB	20.7 MB	5 KB	7 KB	6.5 MB
Bytes Received	9 MB	1.5 MB	248 KB	2 MB	1.7 MB	43 KB	41 KB	7.5 KB
TCP Traffic	7,554	1,546	876	3,645	40,973	110	164	88,937
Connections	30.77	21.57	19.10	80.70	15.46	6.74	12.84	641.61
IPs Visited	22.54	101.41	17.28	21.39	17.01	13.38	12.07	175.48
Countries	1.66	20.73	1.07	2.302	1.65	1.87	0.89	9.63
HTTP	86	47	92	753	7,526	14	26	1,691
GETs	42.49	23.14	16.80	376.72	15.65	7	13.09	845.63
POSTs	0.92	0.54	29.64	3.90	3747.85	0.048	0.001	2.17

Table 6.10: Median values of the features used.

Feature	**Median**							
	Adware	**P2P Bot**	**Clicker**	**Dropper**	**Flooder**	**Miner**	**Ransom**	**Spammer**
Total Traffic	836	512	1,060	767	52,893	394	219	93,564
Outgoing Traffic	256	265	506	312	26,447	81	107	45,071
Incoming Traffic	518	154	478	383	26,406	41	46	48,417
Bytes Sent	19 KB	11 KB	275 KB	18.6 KB	25.8 MB	4.3 KB	4 KB	5.4 MB
Bytes Received	629	66 KB	231 KB	164 KB	2.2 MB	5 KB	16.5 KB	7.3 MB
TCP Traffic	714	232	868	547	52,799	74	84	92,228
Connections	17	18	20	19	14	3	7	530
IPs Visited	18	29	17	18	18	12	11	81
Countries	1	8	1	2	2	2	1	6
HTTP	42	36	97	48	10,562	4	10	120
GETs	20	17.5	20	24	16	1	5	61
POSTs	1	0	31	0	5,240	0	0	0

6.2.3 Results and Discussion

Similarly to before we used Weka to evaluate our system. The Random Forest classifier yielded the highest accuracy and the smallest false-positive rate. A detailed overview

Table 6.11: Detailed accuracy by class.

TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area	Class
0.959	0.072	0.935	0.959	0.947	0.987	Adware
0.93	0.001	0.985	0.93	0.957	0.999	P2P Bot
0.719	0.004	0.745	0.719	0.732	0.904	Clicker
0.835	0.048	0.844	0.835	0.84	0.968	Dropper
0.734	0.002	0.887	0.734	0.803	0.981	Flooder
0.742	0.002	0.885	0.712	0.807	0.936	Coin Miner
0.927	0.009	0.945	0.927	0.936	0.991	Ransomware
0.95	0.001	0.966	0.95	0.958	0.991	Spammer
0.914	0.05	0.913	0.914	0.913	0.981	

of our results using 10-fold cross validation can be found in Table 6.11. Our system achieved an overall TP rate of 91.4% and a FP rate of 5%.

Adware achieved the largest TP rate with 95.9%, while at the same time had the largest FP rate with 7.5%. P2P bots yielded a TP rate of 93% and a FP rate of 1%. The class achieving the lowest TP rate was the clickers with a 71.9% TP rate and 4% FP rate. We believe that this partially stems from the limited number of samples included in the clickers subset. Droppers achieved a relatively high TP rate of 83.5% and a moderate FP rate of 4.8%. The class yielding the second lowest TP rate was the flooders with a TP rate of 73.4%, but at the same time their corresponding FP rate was a mere 2.0%. Coin miners were classified with 74.2% TP rate and 2.0% FP rate. Ransomware yielded high TP and FP rates with 92.7% and 9% respectively. Finally, 91.4% of spammer samples were classified correctly, while a 5% of samples belonging to other classes was classified as such. Table 6.12 constitutes the confusion matrix of the different behaviour types.

Table 6.12: Confusion Matrix.

Adware	P2P Bot	Clicker	Dropper	Flooder	Coin Miner	Ransomware	Spammer	
1966	0	0	76	0	1	6	0	**Adware**
0	132	0	8	0	0	0	1	**P2P Bot**
1	0	41	8	6	0	1	0	**Clicker**
124	1	0	781	0	5	23	1	**Dropper**
1	0	14	2	47	0	0	1	**Flooder**
6	0	0	10	0	46	0	0	**Coin Miner**
3	1	0	38	0	0	532	0	**Ransomware**
1	0	0	2	0	0	0	57	**Spammer**

Overall the the majority of our malware samples were classified correctly. In particular, 95.9% of adware samples were classified appropriately, while a 3.7% of them were misclassified as droppers. In addition, the majority of P2P bots were assigned to their corresponding class, while 5.6% of them were reported as droppers. Furthermore, a

non-negligible proportion of clickers were incorrectly classified, with 14% being classified as droppers and 10% as flooders. Although 83% of droppers were classified correctly, a substantial 13.2% of them were incorrectly classified as adware. Additionally, a notable 21% of flooders were misclassified as clickers.

The classification of coin miners yielded a significant number of false positives (25.6%). In particular, 9.6% of coin miner samples were reported as adware and 16% as droppers. In addition, our system classified appropriately 92.6% of the ransomware samples, while a mere 6.6% of them were assigned to the dropper class. Finally our classifier proved to be remarkably accurate in terms of spammers classification, achieving 95% accuracy, while a negligible number of spammer samples were classified as adware and droppers.

Our results indicate that there exists a small overlap between the classes of adware and droppers. We estimate that the class of droppers has some some unique characteristics. Particularly, as discussed in 2.2.2, the main feature of droppers is that they download other malicious programs to the victim machine. Therefore, we believe the traffic produced by dropper samples strongly depends on the files they download. Intuitively, a dropper downloading a substantially large program produces a larger volume of traffic compared to a sample downloading a smaller file. Apart from that, it is very likely that the new programs get executed as soon as they are downloaded, affecting the nature of the traffic produced. For example if a dropper downloads a malware that falls in the adware category, the new program is very likely to connect to advertising servers. This would affect our classifier that would probably classify the dropper sample as an adware.

We also discern that our system classified some flooders as clickers and vice versa. This verified our estimations that both types produce a significant volume of HTTP traffic and visit a substantial number of web sites. In addition, both classes included a relatively small number of samples, limiting the information incorporated in the classifier.

The limited number of specimens has also affected the classification of coin miners, as the corresponding sub set includes only 62 specimens. However, spammers yielded a remarkable TP rate (95%), despite the limited number of samples included in their sub set. We speculate that this stems from their corresponding feature values as they significantly differ from the rest of our corpus.

Finally, we believe that our system would perform even better, given a larger number of samples for each category. This would result in smaller differences between the cardinality of each sub-set and a more balanced input overall. In addition, an increased number of samples included in each category would assist the system in inferring the intrinsic characteristics of the traffic produced by each class.

Chapter 7

Conclusions and Future Work

The final chapter of this report provides a summary of our system's results and discuss its overall performance. In addition, we discuss potential ways in which we could improve the performance of our system and extend its scope.

7.1 Conclusions

In the context of this thesis, we surveyed the differences in network behaviour between several types of malicious programs. We explicitly segregated malware categories that refer to the spreading mechanism and the self-containment of a program, from the ones describing its underlying behaviour.

Our system attempts to infer the malware type by inspecting network traffic and applying machine learning techniques. Our system consists of two components, the first aims to distinguish the spreading mechanism used by each sample, whereas the latter deals with the challenging task of classifying the specimens with respect to their behaviour. Taking into account the larger number of samples used as well as the substantial number of different families and variants included in each category, we consider our system successful. In particular, our system is capable of inferring the spreading mechanism of malicious programs with 95.1% TP and 4% FP rates. In addition, our system achieved to infer the behaviour of the examined specimens with 91.4% TP and 5% FP rates.

7.2 Future Work

Future work includes testing our system with an increased number of samples. We estimate that a potential increase in the number of samples included in each category would improve even more the accuracy of our system. In addition, we are planning to modify our behaviour classifier by adding new types such as spyware. Finally, it would be very interesting to re-configure our spreading mechanism classifier to consider the class of blended threats, i.e. malware that combine more than one spreading mechanisms, which were out of the scope of this work.

Bibliography

[1] P. S. 2015, "Pandalabs report q1 2015." http://www.pandasecurity.com/mediacenter/src/uploads/2015/05/PandaLabs-Report_Q1-2015.pdf/, 2015. [Online; accessed 20-Jul-2015].

[2] J. Walker. http://www.fourmilab.ch/documents/univac/animal.html, 1996. [Online; accessed 30-Jun-2015].

[3] F. Cohen, "Computer viruses," *Comput. Secur.*, vol. 6, pp. 22–35, Feb. 1987.

[4] E. H. Spafford, "The internet worm program: An analysis," *SIGCOMM Comput. Commun. Rev.*, vol. 19, pp. 17–57, Jan. 1989.

[5] P. Boutin, "Slammed!." http://archive.wired.com/wired/archive/11.07/slammer.html, 2003. [Online; accessed 29-Jun-2015].

[6] R. Naraine, "Storm worm botnet could be world's most powerful supercomputer." http://www.zdnet.com/article/storm-worm-botnet-could-be-worlds-most-powerful-supercomputer/, 2007. [Online; accessed 29-Jun-2015].

[7] B. Neild, "http://edition.cnn.com/2009/tech/ptech/01/16/virus.downadup/?iref=mpstoryview." http://edition.cnn.com/2009/TECH/ptech/01/16/virus.downadup/?iref=mpstoryview, 2009. [Online; accessed 29-Jun-2015].

[8] J. Fildes, "Stuxnet worm 'targeted high-value iranian assets'." http://www.bbc.co.uk/news/technology-11388018, 2010. [Online; accessed 29-Jun-2015].

[9] S. Kramer and J. Bradfield, "A general definition of malware," *Journal in Computer Virology*, vol. 6, no. 2, pp. 105–114, 2010.

[10] D. Guinier, "Biological versus computer viruses," *SIGSAC Rev.*, vol. 7, pp. 1–15, July 1989.

[11] K. K. P. Mell and J. Nusbaum, "Guide to malware incident prevention and handling," tech. rep., NIST, November 2005.

[12] J. Li and P. Knickerbocker, "Functional similarities between computer worms and biological pathogens," *Computers and Security*, vol. 26, no. 4, pp. 338 – 347, 2007.

[13] S. Goel and S. F. Bush, "Biological models of security for virus propagation in computer networks," 2004.

[14] P. Knight, "Iloveyou: Viruses, paranoia, and the environment of risk," *The Sociological Review*, vol. 48, no. S2, pp. 17–30, 2000.

[15] M. Cotton, L. Vegoda, R. Bonica, and B. Haberman, "Special-purpose ip address registries," BCP 153, April 2013.

[16] A. Athanasiou, C. Raftopoulos, E. Thanos, G. Kritharellis, N. Tselikas, I. Foukarakis, and A. Boucouvalas, "Towards privacy-aware target advertising," in *Informatics (PCI), 2012 16th Panhellenic Conference on*, pp. 133–137, Oct 2012.

[17] S. Pilici, "Remove win32/adware.multiplug.h (removal guide)." `http://malwaretips.com/blogs/win32adware-multiplug-h-virus/`, 2013. [Online; accessed 30-Jun-2015].

[18] B. Blunden, *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. USA: Jones and Bartlett Publishers, Inc., 2009.

[19] S. A. Shivale, "Cryptovirology: Virus approach," *CoRR*, vol. abs/1108.2482, 2011.

[20] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," May 2009.

[21] D. Plohmann, E. Gerhards-Padilla, and F. Leder, "Botnets: Measurement, detection, disinfection and defence," tech. rep., Enisa, March 2011.

[22] P. Bächer, T. Holz, M. Kötter, and G. Wicherski, "Know your enemy: Tracking botnets." `https://www.honeynet.org/papers/bots`, 2008. [Online; accessed 15-Jul-2015].

[23] A. Kumar, N. Singh, and D. Desai, "Irc botnets alive, effective and evolving." `http://research.zscaler.com/2015/04/irc-botnets-alive-effective-evolving.html`, 2015. [Online; accessed 15-Jul-2015].

[24] Z. Wang and L. Fu, "The research of detecting irc botnet based on k-means algorithms," in *Communication Systems, Networks and Applications (ICCSNA), 2010 Second International Conference on*, vol. 1, pp. 208–210, June 2010.

[25] K. Chiang and L. Lloyd, "A case study of the rustock rootkit and spam bot," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, (Berkeley, CA, USA), pp. 10–10, USENIX Association, 2007.

[26] N. Daswani and M. Stoppelman, "The anatomy of clickbot.a," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, (Berkeley, CA, USA), pp. 11–11, USENIX Association, 2007.

[27] R. Borgaonkar, "An analysis of the asprox botnet," in *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*, pp. 148–153, July 2010.

[28] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, and N. Borisov, "Stegobot: A covert social network botnet," in *Proceedings of the 13th International Conference on Information Hiding*, IH'11, (Berlin, Heidelberg), pp. 299–313, Springer-Verlag, 2011.

[29] D. Sancho, "Steganography and malware: Concealing code and c&c traffic." `http://blog.trendmicro.com/trendlabs-security-intelligence/steganography-and-malware-concealing-code-and-cc-traffic/`, 2015. [Online; accessed 15-Jul-2015].

[30] A. Crenshaw, "Steganographic command and control: Building a communication channel that withstands hostile scrutiny." `http://www.irongeek.com/i.php?page=security/steganographic-command-and-control/`, 2015. [Online; accessed 15-Jul-2015].

[31] K. Thomas and D. Nicol, "The koobface botnet and the rise of social malware," in *Malicious and Unwanted Software (MALWARE), 2010 5th International Conference on*, pp. 63–70, Oct 2010.

[32] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon, "Peer-to-peer botnets: Overview and case study," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, (Berkeley, CA, USA), pp. 1–1, USENIX Association, 2007.

[33] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm," in *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, LEET'08, (Berkeley, CA, USA), pp. 9:1–9:9, USENIX Association, 2008.

[34] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th Conference on Security Symposium*, SS'08, (Berkeley, CA, USA), pp. 139–154, USENIX Association, 2008.

[35] P. Wood, B. Nahorney, K. Chandrasekar, S. Wallace, and K. Haleym, "Internet security threat report," tech. rep., Symantec, April 2014.

[36] M. Sanford, "Computer viruses and malware by john aycock," *SIGACT News*, vol. 41, pp. 44–47, Mar. 2010.

[37] R. L. Rivest, "The md5 message-digest algorithm," rfc.

[38] D. Eastlake and P. Jones, "Us secure hash algorithm 1 (sha1)," RFC 3174, September 2001.

[39] K. Griffin, S. Schneider, X. Hu, and T.-c. Chiueh, "Automatic generation of string signatures for malware detection," in *Recent Advances in Intrusion Detection* (E. Kirda, S. Jha, and D. Balzarotti, eds.), vol. 5758 of *Lecture Notes in Computer Science*, pp. 101–120, Springer Berlin Heidelberg, 2009.

[40] M. Christiansen, "Bypassing malware defenses," tech. rep., SANS Institute InfoSec Reading Room, May 2010.

[41] N. G. T. D. Forum, "The demise in effectiveness of signature and heuristic based antivirus," tech. rep., NCC Group, 2013.

[42] M. Sikorski and A. Honig, *Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software*. No Starch Press, 2012.

[43] D. Harley and A. Lee, "Heuristic analysis—detecting unknown viruses," tech. rep., Eset, 2009.

[44] F. Nielson, H. R. Nielson, and C. Hankin, *Principles of Program Analysis*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1999.

[45] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna, "Static disassembly of obfuscated binaries," in *In Proceedings of USENIX Security*, pp. 255–270, 2004.

[46] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*, SSYM'03, (Berkeley, CA, USA), pp. 12–12, USENIX Association, 2003.

[47] W. C. Hsieh, D. R. Engler, and G. Back, "Reverse-engineering instruction encodings," in *Proceedings of the General Track: 2001 USENIX Annual Technical Conference*, (Berkeley, CA, USA), pp. 133–145, USENIX Association, 2001.

[48] A. Kiss, J. Jasz, G. Lehotai, and T. Gyimothy, "Interprocedural static slicing of binary executables," in *Source Code Analysis and Manipulation, 2003. Proceedings. Third IEEE International Workshop on*, pp. 118–127, Sept 2003.

[49] P. Anderson, "The use and limitations of static-analysis tools to improve software quality," tech. rep., GrammaTech, 2008.

[50] C. Hekimian, "Limitations of static systems analysis of public policy decisions," in *Technology and Society, 2006. ISTAS 2006. IEEE International Symposium on*, pp. 1–4, June 2006.

[51] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pp. 421–430, Dec 2007.

[52] D. Kirat, G. Vigna, and C. Kruegel, "Barecloud: Bare-metal analysis-based evasive malware detection," in *23rd USENIX Security Symposium (USENIX Security 14)*, (San Diego, CA), pp. 287–301, USENIX Association, Aug. 2014.

[53] A. Mohaisen and O. Alrawi, "Amal: High-fidelity, behavior-based automated malware analysis and classification," in *Information Security Applications* (K.-H. Rhee and J. H. Yi, eds.), vol. 8909 of *Lecture Notes in Computer Science*, pp. 107–121, Springer International Publishing, 2015.

[54] A. Moser, C. Kruegel, and E. Kirda, "Exploring multiple execution paths for malware analysis," in *Security and Privacy, 2007. SP '07. IEEE Symposium on*, pp. 231–245, May 2007.

[55] U. Bayer, C. Kruegel, and E. Kirda, "Ttanalyze: A tool for analyzing malware," 2006.

[56] U. Bayer, A. Moser, C. Kruegel, and E. Kirda, "Dynamic analysis of malicious code," *Journal in Computer Virology*, vol. 2, no. 1, pp. 67–77, 2006.

[57] A. Mohaisen, A. West, A. Mankin, and O. Alrawi, "Chatter: Classifying malware families using system event ordering," in *Communications and Network Security (CNS), 2014 IEEE Conference on*, pp. 283–291, Oct 2014.

[58] X. Chen, J. Andersen, Z. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, pp. 177–186, June 2008.

[59] P. Royal, "Entrapment: Tricking malware with transparent, scalable malware analysis," tech. rep., Georgia Tech Information Security Center, 2012.

[60] J. R. Crandall, G. Wassermann, D. A. S. Oliveira, Z. Su, S. Felix, W. Frederic, and T. Chong, "Temporal search: Detecting hidden malware timebombs with virtual machines," in *Operating Systems Review*, pp. 25–36, ACM Press, 2006.

[61] N. Robillard, "Diffusing a logic bomb," tech. rep., SANS Institute, 2004.

[62] N. Krawetz, "Anti-honeypot technology," *Security Privacy, IEEE*, vol. 2, pp. 76–79, Jan 2004.

[63] A. Kapravelos, M. Cova, C. Kruegel, and G. Vigna, "Escape from monkey island: Evading high-interaction honeyclients," in *Proceedings of the 8th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA'11, (Berlin, Heidelberg), pp. 124–143, Springer-Verlag, 2011.

[64] C. Zou and R. Cunningham, "Honeypot-aware advanced botnet construction and maintenance," in *Dependable Systems and Networks, 2006. DSN 2006. International Conference on*, pp. 199–208, June 2006.

[65] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez, "Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience," in *Security and Privacy, 2006 IEEE Symposium on*, pp. 15 pp.–47, May 2006.

[66] J. Newsome, B. Karp, and D. Song, "Polygraph: automatically generating signatures for polymorphic worms," in *Security and Privacy, 2005 IEEE Symposium on*, pp. 226–241, May 2005.

[67] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, (Berkeley, CA, USA), pp. 4–4, USENIX Association, 2004.

[68] J. Newsome, B. Karp, and D. Song, "Polygraph: automatically generating signatures for polymorphic worms," in *Security and Privacy, 2005 IEEE Symposium on*, pp. 226–241, May 2005.

[69] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "Bothunter: Detecting malware infection through ids-driven dialog correlation," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, SS'07, (Berkeley, CA, USA), pp. 12:1–12:16, USENIX Association, 2007.

[70] G. Gu, J. Zhang, and W. Lee, "Botsniffer: Detecting botnet command and control channels in network traffic.," in *NDSS*, The Internet Society, 2008.

[71] G. Stringhini, T. Holz, B. Stone-Gross, C. Kruegel, and G. Vigna, "Botmagnifier: Locating spambots on the internet.," in *USENIX Security Symposium*, USENIX Association, 2011.

[72] R. Perdisci, W. Lee, and N. Feamster, "Behavioral clustering of http-based malware and signature generation using malicious network traces," in *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, (Berkeley, CA, USA), pp. 26–26, USENIX Association, 2010.

[73] Y. Park, D. Reeves, V. Mulukutla, and B. Sundaravel, "Fast malware classification by automated behavioral graph matching," in *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, CSIIRW '10, (New York, NY, USA), pp. 45:1–45:4, ACM, 2010.

[74] K. Rieck, T. Holz, C. Willems, P. Dussel, and P. Laskov, "Learning and classification of malware behavior," in *Detection of Intrusions and Malware, and Vulnerability Assessment* (D. Zamboni, ed.), vol. 5137 of *Lecture Notes in Computer Science*, pp. 108–125, Springer Berlin Heidelberg, 2008.

[75] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *J. Comput. Secur.*, vol. 19, pp. 639–668, Dec. 2011.

[76] H. Zhao, M. Xu, N. Zheng, J. Yao, and Q. Ho, "Malicious executables classification based on behavioral factor analysis," in *e-Education, e-Business, e-Management, and e-Learning, 2010. IC4E '10. International Conference on*, pp. 502–506, Jan 2010.

[77] G. Yan, N. Brown, and D. Kong, "Exploring discriminatory features for automated malware classification," in *Detection of Intrusions and Malware, and Vulnerability Assessment* (K. Rieck, P. Stewin, and J.-P. Seifert, eds.), vol. 7967 of *Lecture Notes in Computer Science*, pp. 41–61, Springer Berlin Heidelberg, 2013.

[78] A. Shabtai, R. Moskovitch, Y. Elovici, and C. Glezer, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," *Inf. Secur. Tech. Rep.*, vol. 14, pp. 16–29, Feb. 2009.

[79] D. Kong and G. Yan, "Discriminant malware distance learning on structuralinformation for automated malware classification," in *Proceedings of the ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '13, (New York, NY, USA), pp. 347–348, ACM, 2013.

[80] J. Z. Kolter and M. A. Maloof, "Learning to detect and classify malicious executables in the wild," *J. Mach. Learn. Res.*, vol. 7, pp. 2721–2744, Dec. 2006.

[81] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 2009)*, 1 2009.

[82] M. Zhang. `http://petapixel.com/2015/05/29/sourceforge-accused-of-bundling-gimp-with-adware/`, 2015. [Online; accessed 20-Jul-2015].

[83] S. Gallagher. `http://arstechnica.co.uk/information-technology/2015/05/sourceforge-grabs-gimp-for-windows-account-wraps-installer-in-bundle-pushing-adware/`, 2015. [Online; accessed 20-Jul-2015].

[84] A. K. Jain, M. N. Murty, and P. J. Flynn, "Data clustering: A review," *ACM Comput. Surv.*, vol. 31, pp. 264–323, Sept. 1999.

[85] A. M. Dan Pelleg, "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning*, (San Francisco), pp. 727–734, Morgan Kaufmann, 2000.

[86] C. Rossow, C. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. van Steen, "Prudent practices for designing malware experiments: Status quo and outlook," in *Security and Privacy (SP), 2012 IEEE Symposium on*, pp. 65–79, May 2012.

[87] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario, "Automated classification and analysis of internet malware," in *Recent Advances in Intrusion Detection* (C. Kruegel, R. Lippmann, and A. Clark, eds.), vol. 4637 of *Lecture Notes in Computer Science*, pp. 178–197, Springer Berlin Heidelberg, 2007.

[88] C. E. Shannon, "A mathematical theory of communication," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 5, pp. 3–55, Jan. 2001.

[89] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[90] W. Strayer, R. Walsh, C. Livadas, and D. Lapsley, "Detecting botnets with tight command and control," in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pp. 195–202, Nov 2006.

[91] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale botnet detection and characterization," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, (Berkeley, CA, USA), pp. 7–7, USENIX Association, 2007.

[92] T.-F. Yen and M. K. Reiter, "Traffic aggregation for malware detection," in *Proceedings of the 5th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, (Berlin, Heidelberg), pp. 207–227, Springer-Verlag, 2008.

[93] J. R. Binkley and S. Singh, "An algorithm for anomaly-based botnet detection," in *Proceedings of the 2Nd Conference on Steps to Reducing Unwanted Traffic on the Internet - Volume 2*, SRUTI'06, (Berkeley, CA, USA), pp. 7–7, USENIX Association, 2006.

[94] J. Goebel and T. Holz, "Rishi: Identify bot contaminated hosts by irc nickname evaluation," in *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, (Berkeley, CA, USA), pp. 8–8, USENIX Association, 2007.

[95] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX Conference on System Administration*, LISA '99, (Berkeley, CA, USA), pp. 229–238, USENIX Association, 1999.

[96] F. Bellard, "Qemu, a fast and portable dynamic translator," in *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, (Berkeley, CA, USA), pp. 41–41, USENIX Association, 2005.

[97] C. Lattner and V. Adve, "Llvm: A compilation framework for lifelong program analysis & transformation," in *Proceedings of the International Symposium on Code Generation and Optimization: Feedback-directed and Runtime Optimization*, CGO '04, (Washington, DC, USA), pp. 75–, IEEE Computer Society, 2004.

[98] D. Scocco. `http://www.dailyblogtips.com/on-domain-names-size-and-quality-does-matter//`, 2008. [Online; accessed 29-Jul-2015].