Detecting botnets using file system indicators

Master's thesis

University of Twente

Author:
Peter Wagenaar

Committee members:
Prof. Dr. Pieter H. Hartel
Dr. Damiano Bolzoni
Frank Bernaards LLM (NHTCU)

December 12, 2012

Abstract

Botnets, large groups of networked zombie computers under centralised control, are recognised as one of the major threats on the internet. There is a lot of research towards ways of detecting botnets, in particular towards detecting Command and Control servers. Most of the research is focused on trying to detect the commands that these servers send to the bots over the network. For this research, we have looked at botnets from a botmaster's perspective. First, we characterise several botnet enhancing techniques using three aspects: resilience, stealth and churn. We see that these enhancements are usually employed in the network communications between the C&C and the bots. This leads us to our second contribution: we propose a new botnet detection method based on the way C&C's are present on the file system. We define a set of file system based indicators and use them to search for C&C's in images of hard disks. We investigate how the aspects resilience, stealth and churn apply to each of the indicators and discuss countermeasures botmasters could take to evade detection. We validate our method by applying it to a test dataset of 94 disk images, 16 of which contain C&C installations, and show that low false positive and false negative ratio's can be achieved. Approaching the botnet detection problem from this angle is novel, which provides a basis for further research.

Contents

Acknowledgements

This work would not have been possible without the guidance, help and feedback of several individuals. I would like to take this opportunity to express my sincere gratitude towards them.

First of all, I would like to thank Prof. Dr. P.H. Hartel for providing me with the opportunity to perform this research, for giving me the freedom to find my own way, for helping me enthusiastically throughout the project and for keeping me on the right track with his critical feedback.

I wish to thank F. Bernaards, LLM and E. Monsma, MSc. for welcoming me to the world of crime fighting and for being the spiders in the web of all the external contacts that helped realize this research.

A thank you to Dr. D. Bolzoni for making time in his busy schedule to be part of my graduation committee and for technically reviewing my work.

I am also grateful to the members of the National High Tech Crime unit of the Netherlands' Police Agency for lending me their technical expertise and creativity in the process of developing the botnet detection method described in this thesis. Besides, they have really made me feel at home, for which I am sincerely grateful.

I wish to thank P. Takkenberg in particular for giving me the time to finish my thesis even though other matters were more pressing.

Last but not least, I would like to thank my family and my partner for providing me with the energy needed for this research and for giving me a place to be myself.

Chapter 1

Introduction

Botnets, large groups of networked zombie computers under centralised control, are recognised as one of the major threats on the internet. As a proper understanding of this phenomenon is vital to be able to fight it, a lot of research is dedicated to this topic. Academia as well as anti virus companies, operating system developers and law enforcement agencies all spend considerable time and effort in botnet research, botnet disruption and botnet takedown operations.

A large part of the academic botnet research is geared towards developing ways of detecting botnets. Almost all detection methods discussed in the literature focus on network traffic, be it analysis of individual packets or by looking at more general netflow data. This focus makes sense from the perspective of researchers (academic as well as law enforcement agencies). Network data can be monitored and analyzed with relative ease: legal systems usually have provisions for intercepting network traffic and there are enough tools available for the analysis of intercepted data. The control traffic is also what separates botnet bots from, for instance, regular viruses and trojan horses, making it an obvious candidate for botnet detection.

1.1 Problem

What happens, however, when network data is unavailable? Scenario's for such a case are the following:

- If a number of different servers share the same internet connection it may be impossible to determine which of the servers is the actual C&C and which are less interesting.
- When a number of servers is confiscated for an offense other than hosting botnet infrastructure, there may not be enough legal reason to capture network traffic for every server, or not enough capacity to do so, rendering network based botnet detection impossible. Such a scenario may occur when dealing with bulletproof hosting providers (see next chapter), who rent out a number of servers, each of which can be used for a different (illicit) purpose.
- Automatically detecting which servers host a botnet C&C may also be used in a triage procedure, to reduce the number of servers that one has to investigate by hand.
- A final use case is botnet detection on live systems, where a hosting company could periodically scan whether rented systems are being used as botnet C&C's, possibly alongside traditional network based botnet detection.

In these scenario's it may help to have other C&C detection methods available, methods that rely less on network traffic and more on the servers themselves. Our problem statement is as follows:

How can we determine whether a given system contains a botnet C&C, without the use of network data?

There is little research, however, on the layout and used technologies of the C&C soft-

ware. This can be explained in part by the difficulty of gaining access to C&C software: apart from a few commercial crimeware kits (such as ZeuS[52] and SpyEye[53]) generally only law enforcement agencies have access to C&C software through confiscated C&C machines.

1.2 Contribution

For our research, we will look at botnets from the botmaster's perspective. Our contribution is twofold. We will analyse a number of botnet enhancing techniques in light of three aspects: resilience, stealth and churn. Resilience defines how well a botnet is equipped versus takedown. Stealth defines how well a botnet can operate without being detected. Churn defines how fast bots join the botnet, i.e how fast machines are infected and how fast infected machines are disinfected again. These aspects help us look at botnets from a botmaster's perspective, so that we can see the motivation behind using certain botnet enhancing techniques.

For our second contribution, we propose a new botnet detection method.

We will focus on the way a C&C is stored on a file system. Information on the file system is well structured, can be accessed and processed quickly and contains the heart of the C&C infrastructure: the place where the actual C&C program code resides. We will try to find a set of indicators that, when searched for within a file system, can indicate whether a C&C infrastructure is present on the machine or not.

We will develop a proof of concept botnet detection implementation which, using these indicators, can search for botnet C&C infrastructures in disk images. We will apply our method to a test dataset consisting of 94 disk images of confiscated servers of a bulletproof hosting provider and discuss the results for each indicator.

1.3 Research questions

The research questions to which we try to find an answer are the following:

- *Which file system indicators can we use to correctly identify the presence of a botnet C&C?*
- *Which indicator has the lowest false positive rate and which has the lowest false negative rate and why?*
- *How feasible is using these indicators when searching through large datasets?*

In our search for file system indicators, we focus on the "quick wins": simple yet fast and effective methods of finding out whether a system hosts a C&C infrastructure or not. The reasoning is that simple indicators can be located and processed fast so that many systems can be checked as fast as possible. Our research is highly exploratory: we think of some indicators, build a proof of concept implementation that will search for these indicators and see how each indicator performs. Some indicators may give a high false positive or false negative ratio, some will work unexpectedly well. Since this is one of the first researches of its kind, we just want to see what's out there and provide a basis for further research.

The remainder of this work is as follows. In the next chapter we will give an overview of the current literature regarding botnet characteristics and botnet detection methods. After that, in chapter 3, we will look at our own detection method and at the setup for the experiment we conduct. Then, in chapter 4, we discuss the results of our experiments. We conclude in chapter 5. The appendix at the end contains some metrics and other information about our proof of concept implementation, including some possible optimizations.

Chapter 2

Related work

This chapter provides an overview of the current state of the literature regarding botnet characteristics and botnet detection methods. The contents are as follows. We will start with describing what botnets are, how they work and what they are used for in section 2.1. Then, in section 2.2, we will dive deeper into the characteristics of different botnet designs. We will look at the different communication channels used and at some of the techniques to enhance botnets. After that, in section 2.3 we will discuss several detection methodologies, subdividing them into those based on malicious traffic and those based on C&C traffic.

2.1 Botnet overview

Offenders have always been resourceful in their ways of making money. As people incorporate internet in more and more aspects of their life, offenders started seeing opportunities for making money in the digital world through scams, extortion schemes, property destruction, theft, etc. One branch of this so-called "cybercrime" involves the use of malicious pieces of software, called "malware", with the goal of making a target's computer perform actions that the target hasn't consented to or knows about. If the malware allows the offender to control the infected computers remotely, such a group of computers is generally called a "botnet". A single computer within this group is known as a "bot" or "zombie" and the person(s) controlling the computers are known as "botherders" or "botmasters".

Besides offenders there are other parties that use botnets, such as (cyber)terrorists[54] and government agencies[55]. Although we will mention the activities they employ below, the setups they use are largely outside the scope of this paper and will not be discussed further. Before we look at botnet activities, however, we will first discuss botnet design, to give the reader a better understanding of what botnets are and how they operate.

2.1.1 Botnet design

Although there are many different possible designs for botnets, all designs have some factors in common. First of all, all botnets consist of a group of devices infected with malware to perform the actual work. These devices can be infected through a variety of attacks including drive-by downloads, automated exploitation (like with computer worms), malware on USB flash drives or malicious attachments in e-mails. The goal of the infection is to get a piece of malware running on the target's device, preferably without the target knowing about it; this way the target is less likely to take steps to remove the malware.

Once a target's device is infected, the botmaster needs to set up some sort of control mechanism to tell the infected device what to do. The way this control mechanism is designed forms one of the characterizing features used by researchers to taxonomize botnets[1][2][3].

On one side of the spectrum there is a centralized control architecture. In this architecture all the bots connect to one or more centrally located servers, called "Command & Control servers" (also known as C&C or C2). The botmaster connects to the C&C as well and the

commands which she provides get distributed from this central point to all the bots.

On the other side there is a decentralized architecture. In this architecture there are no centralized machines; every machine in the infrastructure is of equal importance. Commands are given to one or a few bots, chosen at random, which in turn make sure that the commands reach all the other bots in the botnet.

These two architectures form two extremes. In reality botnets employ aspects of each architecture in a hybrid design. We will discuss this more deeply in section 2.2.

Reported sizes of botnets vary greatly. Research into enumerating botnets is still ongoing[4][5] and each new trick botmasters invent to hide their botnets makes estimating the actual size harder. There have been reports of botnets with over 30 million total infections (Bredolab, see [56]). Most of the malicious activities (see below) hinge on the number of bots available, so maximizing the number of infections is paramount for the botmaster.

2.1.2 Malicious botnet activities

Botmasters use their botnets for all kinds of different purposes, most of which are considered malicious and/or harmful. The reasons botmasters use botnets for these activities are many. As we will see in the next section, botnets provide a measure of anonymity which, because of the malicious nature of the activities, is a necessity. Furthermore, the geographically distributed nature of botnets means that it is difficult for researchers to track them. Different time zones, languages, laws and jurisdictions all form barriers, aiding the botmaster in staying ahead of the law[6].

Some activities that have been seen in the field include:

Spam[7]

Sending spam means surreptitiously sending unsolicited e-mails in the hope that the recipient follows the link inside the e-mail. The page the link points to can, for instance, be a phishing page (designed to steal login credentials) or a pharmacy scam (designed to sell counterfeit drugs). Having more bots in the botnet means that more mails can be sent, thus raising the chances of a recipient following the link.

DDoS[8]

DDoS attacks are attacks on a website or other internet facing service, performed by generating so much traffic to it that it can't handle all the requests and will be unavailable for regular users. This can then be used either in an extortion scam ("pay me 100.000$ or your site will remain offline") or purely as a way of harassing others. Having more bots in the botnet means that more traffic can be generated, which in turn means that larger, more popular websites can be taken offline. This can be useful in an extortion scheme, where more money can be extorted for larger websites.

Information stealing[57]

Some offenders gather login credentials, e-mail addresses, company secrets, credit card numbers or other information using their botnet. The information can be gathered by various methods, such as keyloggers or file system crawlers. It can be sold on the black market or used by the botmaster herself for other purposes. Obviously, having more bots means more gathered information.

Pay-per-install[9]

Already infected machines can be sold to other botmasters so that they can install their own malware in an easy fashion. In such a

pay-per-install scheme, the other botmaster doesn't have to devise a way of spreading the malware on its own. Having more bots in the pay-per-install botnet means that more "installs" can be sold.

Click Fraud[10]

A botmaster can also use his botnet to generate clicks on his own website's advertisements, with the goal of getting paid for each click. If there are More bots in the botnet, this means more clicks and thus more money.

Mining or stealing Bitcoins[58][59]

Mining or stealing bitcoins is a relatively new botnet activity. Bitcoins[60] are a digital currency based on cryptographic principles that started emerging around 2009 and has grown to quite a big user base[61]. The idea behind Bitcoins is that they can be "mined" (basically generated) on a computer and that they can be used in anonymous but verifiable transactions to purchase goods. Bitcoin mining requires significant computational resources; the more resources one has available the quicker one can generate Bitcoins. The massive computational power a botnet provides can thus be used to mine Bitcoins: having more bots means more computational power and thus more Bitcoins. Of course, one can also try to steal other people's Bitcoins as they are stored in a file (called a "wallet") on the computer.

Cyberwarfare[11]

Malware can be used by political groups or nations to attack another political group or nation, with the intention of causing damage or disruption to the target's systems, or to steal information. This activity was first seen with the Stuxnet virus which was allegedly designed by the United States of America together with Israel to impair Iran's uranium enrichment capabilities. Another example is the Flame virus, which sends screenshots, keyboard activity, network traffic and documents from the infected computer to its C&C. Flame infections were initially only found at government organizations and prominent state officials in Middle Eastern countries, leading the belief that it was indeed an act of cyberwarfare.

Both these examples were targeted attacks, instead of the erratic, non discriminant infection behaviour we see with "regular" botnets. Nonetheless, we classify cyberwarfare as a botnet activity because of other properties: both viruses were designed to spread to other systems and both were controlled from a central location. As the attacks were targeted and never meant to leak out to the public, this is one of the few botnet activities in which larger numbers aren't directly beneficial to the botmaster.

Botnet improvement

Using the bots in the botnet to enhance the botnet itself by raising botnet resilience, stealth or churn (see section 2.2). The exact methods will be explained later, but this usually means using the bots as a "shield" of sorts, to make sure that researchers are unable to find the real identity of the botmaster.

Hacktivism[62]

Even though hacktivism isn't strictly a botnet activity, it is worth mentioning here. Most of the activities mentioned above can also be used from an idealogical instead of a monetary standpoint. Instead of trying to make money, people use botnets to get a political standpoint across. These protests often take the form of so-called opt-in DDoS botnets. These botnets are formed by people who willingly install a piece of software on their system after which their system can join in on a DDoS attack. These attacks are usually targeted against websites that publicize a politi-

cal opinion different from the attackers' opinion. An example of such an opt-in DDoS botnet is the attacks by hacker group Anonymous on the website of the Scientology church, in which people could install the LOIC[1] tool to join in on the DDoS attack[63].

2.1.3 Comparing botnets to regular cloud services

Parallels can be drawn between botnets and regular cloud services, such as Amazon's EC2. Both provide massive computational power, network bandwidth and storage. Some botnets provide lease services, where one can hire part of the botnet to perform tasks such as the ones mentioned above, but also for use as a regular hosting service or computation platform[64][65]. Botnets and cloud services also provide a level of resilience: a service that runs on top of either of the two infrastructures can remain online even if some of the nodes that make up the infrastructure get taken down.

The three botnet aspects that we will discuss in the next section, however, reveal some differences as well. First of all, because of the illegitimate nature of botnets they tend to have a stealthy setup, trying to hide as much of the infrastructure as possible from the outside world. Secondly, the composition of nodes in a botnet changes more rapidly than in a regular cloud service. Regular cloud services usually only swap out nodes if they suffer hardware failure and add nodes when the computational requirements surpass the computational power available. The involuntary participation of the infected machines in a botnet means that the number of node joins and node leaves is much higher than in a regular cloud service. A final difference is the number of illegitimate services ran from a botnet; a regular cloud service has strict rules on what activities are allowed and which are not whereas a botnet (even a rented one) can be used for any and all activities.

In some ways, cloud service providers could learn from the evolution of botnets. Methods for decentralized control, dealing with node failure and dealing with the heterogeneity of the operating systems of different nodes are all problems shared between cloud service providers and botmasters.

After this brief overview of the botnet world we will now discuss the characteristics of botnets over the years.

2.2 Botnet characteristics

Today's botnets have come a long way from the IRC based bots that were used at the beginning of the 21st century. In this chapter we will provide an overview of historic and current botnet characteristics. We will discuss these characteristics in light of three main botnet aspects: resilience, stealth and churn.

Resilience indicates how well a botnet can withstand takedown. Setting up a botnet takes time and money, so keeping the botnet online as long as possible raises the botmaster's return on investment.

Stealth means how well a botnet can stay hidden from researchers and targets. The longer a botnet stays hidden, the longer it can operate without problems. Stealth also indicates how well the botmaster can stay hidden in case of an investigation.

Churn has everything to do with the spreading capability of the malware used to create the botnet. It indicates how fast new machines are infected and how fast malware is removed again. As long as the churn is "positive", more bots are joining than there are bots leaving, meaning that the botnet grows in size.

[1] Low Orbit Ion Cannon

These aspects impact the size and availability of a botnet: a botmaster wants to have a botnet with many bots that can operate undisturbed for as long as possible. A high positive churn impacts the botnet's size because churn defines how fast bots join the botnet. A high resilience and good stealth helps to let the botnet operate undisturbed.

We have chosen these three aspects as they show what kind of properties botmasters want in their botnets. Compared to the survey work by Bailey *et al.* [12] and Li *et al.* [13], our aspects provide a view on the characteristics of botnets that is centralised around the botmaster's wishes, instead of focusing purely on the botnet characteristics from an outsider's perspective.

The following sections will discuss some of the techniques employed by botmasters to enhance their botnets and we will show how each technique impacts resilience, stealth and churn. As each of the techniques described below requires effort to implement, choosing whether or not to add a given technique to a botnet will always be a matter of weighing the implementation costs versus the benefits that the technique will bring.

2.2.1 Attack vectors

An important factor in the success of a botnet is its ability to infect new machines. As we have seen in section 2 there are many possible attack vectors, each with its own advantages and disadvantages. Which attack vector(s) the botmaster employs greatly impacts resilience, stealth and churn. For instance, if she chooses to use a virus that exploits a known vulnerability in some piece of software, it is likely that the success rate won't be very high, as targets have already patched the vulnerability. If she develops a virus that exploits an unknown vulnerability, however, the chance of infection will become higher, but developing such a virus takes a lot more time. Besides viruses that directly attack software she could also try to use phishing e-mails to get a target to start the malware or use malware that spreads using USB sticks only. The attack vector's infection rate determines how fast bots join the botnet and thus has a great impact on botnet churn.

A second factor that impacts botnet churn is the speed with which an infection can be removed from a system. If the malware targets a piece of software that is developed by a company that hardly ever fixes vulnerabilities in their software, targets are going to have a hard time protecting their system from infections. Furthermore, the employed malware could use rootkit and other process hiding techniques, hiding the malware from the operating system, making removal a difficult task. Increasing the difficulty with which the malware can be removed increases the botnet churn rate and raises botnet resilience.

The aggressiveness of the chosen attack vector also determines how stealthy the botnet is going to be: if 100.000 computers are infected within a few hours, somebody is bound to notice whereas a slow infection using USB sticks only may go undetected for quite some time.

2.2.2 Overall botnet infrastructure and communication channels

Once machines are infected, they need to receive commands from the C&C. Over the years a number of different protocols have been used by botmasters to communicate with their bots. The botnet business started with bots controlled through IRC in a purely centralized fashion. Soon after, botmasters realized that IRC-based botnets were too easy to takedown, so they moved to other channels instead, such as botnets controlled with an HTTP webserver. Lately, even more advanced communication channels have begun to emerge, such as peer-to-peer channels and even misuse of social networking sites such as

Facebook or Twitter. In this section we will look at some of these communication channels and their impact on the three main botnet aspects.

All the different communication channels mentioned below can be used to add stealth and resilience to the botnet. A botnet investigation usually starts with a researcher investigating an infected machine and trying to trace control communications from this machine back to the botmaster. Adding one or more proxy layers between the botmaster and her bots means that finding the botmaster becomes more difficult, thus adding stealth. Furthermore, controlling the bots using multiple C&C machines in parallel means that when a single C&C machine is taken down, the botnet can still survive, thus raising resilience.

IRC based botnets

One of the first programs with botnet characteristics known to the academic world was Eggdrop[66], an IRC bot used to manage and protect IRC channels from takeover attempts[13]. Although Eggdrop does not have malicious capabilities per se, it was one of the first to make multiple programs work together towards a certain goal under centralized remote management. Furthermore, we see that IRC is the communications channel of choice for botmasters at the start of the botnet era. Various bots using IRC as the primary communication channel have been developed since Eggdrop, including Slapper, Agobot and Sdbot, the latter two of which are still actively seen in the wild[67].

Just like regular IRC users, bots of IRC based botnets connect to a centralized IRC server. The botmaster controls the bots by connecting to the same IRC network and then either gives the bots commands one by one, or lets all the bots join a single chatroom in which the botmaster posts commands for all bots to see. The botmaster can use a public IRC server or run her own. Both possibilities are encountered in the field; a public server requires less setup but running one's own server provides slightly more anonymity. A third party server usually logs all incoming connections, including the botmaster's IP, meaning that a takedown operation could uncover the true identity of the botmaster[68].

Most botmasters use IRC as a control channel because of its simplicity. Using IRC does not greatly increase resilience, stealth or churn, but is easy to setup, easy to maintain and provides the most basic form of centralised control. This is probably also the reason why IRC-based botnets are still seen in the wild, even though there are plenty of detection methods available for IRC-based botnets (see section 2.3).

Web (HTTP) based botnets

In recent years, more and more botnets have appeared that use communication channels other than IRC[69]. The reason for this is probably that it is not very hard to detect IRC based botnets in network traffic, as proven by the general interest and the amount of successful detection methods provided by the academic world[70][68][71]. To remain beyond the reach of the law, botmasters need to constantly tweak, upgrade and reinvent their botnet architectures and corresponding botnet communication channels. Two of these new channels are web-based protocols (e.g. HTTP) and peer-to-peer (P2P) protocols.

Many of today's large[2] botnets use HTTP as a communication channel: Bredolab (2009)[72], Rustock (2006)[14], Conficker (2008)[73], TDL4 (2010)[64] and ZeuS (2007?)[15] are prominent examples. Using HTTP has a few

[2]There is a lot of research on how to measure a botnet's size, here we mean the botnets with the most overall infections

advantages over IRC as a communications channel[16]:

- Firewalls usually allow free access to the web by default, whereas they may block IRC traffic. This way, more infected computers can reach the C&C, effectively enlarging the botnet.
- There is a large amount of "regular" web traffic, making hiding in the crowd easy. This adds stealth to the botnet.
- Extra anonymity can be gained by using a hacked webserver instead of one's own machines; there are enough insecure webservers out there. This, again, raises stealth.
- Encryption of the traffic between a bot and the C&C can be implemented easily using HTTPS. This raises stealth, but also raises resilience as researchers will have a harder time figuring out how a botnet works.

In contrast with IRC based bots, web based bots usually work in a pull fashion; commands are put online on publicly facing web servers which the bots occasionally poll to see if there are new commands.

Peer-to-peer botnets

Recently (starting at about 2003) we started seeing a third type of communication channel become popular besides HTTP: peer-to-peer based communication channels. Starting with the Slapper worm[17] and later the Nugache and Storm[18] botnets, application of P2P technology promises to substantially change the botnet world.

Both IRC and Web based botnets have a purely centralized control architecture. That is, the botmaster has all the bots communicate with one or more centralized servers to receive their commands. These centralized servers form a weak point in the botnet infrastructure; if a law enforcement agency or other botmaster somehow takes these servers down the botnet is crippled or even completely shut down since the bots can no longer be controlled.

P2P techniques can be used to mitigate this problem[19]. In a P2P botnet, bots have the ability to contact each other directly, passing on commands and data from one bot to another. To give a command to her botnet, a botmaster contacts a few nodes and gives them the command. These bots will, in turn, make sure that the command is passed on to every node that they know. The receiving nodes will, in turn, propagate the command to all the nodes they know, etc. In this way the command ripples through the entire botnet, provided that every node in the botnet can be reached from the first few (i.e. the network that the bots form amongst themselves is a connected graph). This simple, flat architecture is employed by the Slapper botnet [17].

Another P2P architecture (such as used by Nugache) "promote" bots to become so-called servant bots, putting them hierarchically above a group of regular bots. From that moment on, the servant bot is responsible for relaying commands to these regular bots. If the servant bot goes offline for whatever reason, another servant bot is chosen to take its place. Commands can propagate between the servants in a P2P fashion (one servant notifies another) or again in a hierarchical fashion where a "super-servant" notifies all other servants. In this hierarchical architecture, a bot does not need to know about all the other bots in the botnet; it suffices if a bot knows those bots that come "below" it and those that are "above" it.

The advantage of using such a dynamic hierarchical architecture is that command latency is much lower than in the normal flat design. Since the flat design has no guaran-

tees on which bot knows which other bots, commands can take a long time to traverse the entire network [20]. The hierarchical design helps mitigate this issue by making sure the command travels between the servant bots first, which amongst each other know all the bots in the network.

Note that the absence of a centralized command infrastructure is not a necessity for a botnet to be classified as a P2P botnet; Storm is a good example of a "hybrid P2P botnet", which mixes a P2P structure with centralized command and data store components[74].

With regard to resilience and stealth, using P2P provides some additional advantages over using HTTP. First of all, since the botnet itself provides the C&C machines it is easy to replace C&C machines that are taken down. In an HTTP scheme replacement C&C machines have to be bought or hacked, meaning a lot of work for the botmaster. Secondly, periodically changing which machines perform C&C functions is easy. This provides extra stealth and resilience to the botnet, as suspicious C&C activities can only be monitored for a short while on the same machine, after which the channel switches again.

2.2.3 Using DNS

There are several other techniques botmasters use to enhance their botnets. In this section we will look at a few of them. The first three of these revolve around the use of DNS and they form three steps of increasing resilience. One could argue that these techniques also increase stealth, as they add a layer of indirection between the bots and the C&C, but since regular DNS has to be publicly accessible it is also accessible to researchers, meaning that no additional stealth is provided.

The first and simplest technique involving DNS is to let bots find the C&C by letting them resolve a domain instead of hardcoding an IP address. If (part of) the C&C is taken down by authorities, the botmaster can change the DNS record to point to a newly set up C&C infrastructure with a different IP address, effectively keeping the botnet alive with minimal losses. Using low TTL values for the DNS record ensures that bots always have the latest IP address to connect to.

Sinkholing and domain-flux

One of the techniques researchers use to analyse and disrupt botnets is called sinkholing. In general network terms, sinkholing means routing all traffic with a certain destination away from its original destination and towards a new destination. The objective for the researcher is to make sure that instead of the real C&C, the bots in a botnet either can't talk to a server at all (blackholing) or talk to a server under the researcher's control. The former technique only disrupts the botnet where the latter gives the researcher full control, so he could use it to collect data or even remove the bot software from the infected machines, effectively taking down the whole botnet.

There are three ways of sinkholing[21]:

1. Redirect all traffic matching a certain destination domain name to the researchers machine (DNS based)
2. Redirect all traffic matching a certain destination IP to the researchers machine (IP based)
3. Rebind the IP of the destination machine to the researchers machine (IP based)

The first measure has to be implemented at the registrar which handles the registration of the DNS records in question. The other two measures can only be implemented at the ISP which hosts the C&C infrastructure.

Botmasters use several approaches to mitigate the effects of sinkholing. First of all, most

botmasters set up their C&C infrastructures at so called "bulletproof hosting providers"; providers which do not respond to complaints from outside parties. As the 2nd and 3rd sinkhole methods require ISP coöperation, using a bulletproof hosting provider makes it difficult for researchers to harm the botnet using IP based sinkholing.

A simple answer to the DNS based sinkhole method is to design the botnet in such a way that no DNS is required at all, as we see with the Nugache botnet[18]. Besides this, the botmaster can also make her bots try not just a single domain but a large set of domain names when trying to find their C&C. In this way, researchers will have to sinkhole a large number of domains, increasing the difficulty of the operation. A further enhancement to this scheme is to let the bots create the list of domain names anew periodically, based on some seed (such as the current date). This technique is called "domain-flux"[75]. Since the botmaster knows the algorithm and the seed that the bots will use to generate the domain names she can calculate a list of domains in advance, register one of them beforehand and let it resolve to the C&C server of her choice. If researchers want to sinkhole a botnet using domain-flux techniques, they would need to know which domains are going to be used for a given period (i.e. by reversing the bot binary), then register all those domain names before the botmaster does. If the effort succeeds the researchers have control of the botnet for as long as the bots use the same domain name, since the next period all the bots will generate a new list of domains. To keep control of the botnet the researchers would either have to keep registering all domain names for every period to come, or they would have to change how the bot generates domain names, a procedure raising legal and ethical questions because the researcher would have to use the botnet to change the bot software.

Fast-Flux

Many of the uses of botnets require an internet facing webserver to distribute malicious content: hosting phishing sites, providing bots with updates, providing spam sending bots with new e-mail addresses and templates and sites where password stealing bots can dump their gathered information (see also section 2 on the uses of botnets). This proves to be a problem for the botmaster. If a static location (such as a rented webserver or a hacked webserver of a third party) is used to host the malicious site on, researchers can find the IP address of this static location with relative ease; they can simply resolve the domain name found in the phishing e-mail or look at the network traffic of an infected machine to see where it connects to. To prevent researchers from finding the IP address in such an easy manner and to make taking down these hosting sites harder, botmasters can use another dynamic DNS technique known as "Fast-Flux"[22]. In a Fast-Flux scheme the botmaster makes sure that a malicious domain does not directly resolve to the IP address of the machine hosting the malicious content, but instead she lets it resolve to a subset of all the IP addresses of bots in her botnet. Since the bot population in a botnet changes rapidly (bots go offline, bots come online, new machines are infected, machines are disinfected by anti virus software, etc.), the set of IP addresses that the domain resolves to needs to change rapidly as well. The botmaster could even decide to switch out certain IP addresses based on bot health; bots with high bandwidth and low latency may be preferable over other bots. Using a Fast-Flux scheme, the botmaster can use the bots themselves to host the content, making it harder for researchers to take down the hosting facilities.

An improvement to the Fast-Flux scheme is to combine it with a reverse proxy scheme. Instead of hosting the content on each bot par-

ticipating in the Fast-Flux, the content can be hosted on a single server, called the Fast-Flux "Mothership" in the literature. Each bot can then act as a reverse proxy, transparently forwarding requests for content to the Mothership and forwarding every reply from the Mothership back to the target. This technique is called a "reverse proxy" because the bots function as a proxy for a group of servers, whereas a normal "forward proxy" functions as a proxy for a group of clients.

Another improvement is to not just use Fast-Flux for content, but also for DNS itself; besides hosting content every bot can become a DNS server, accepting and forwarding DNS queries, providing another layer of redirection. In the literature this scheme is known as "double-flux" whereas the simpeler scheme is known as "single-flux"[76].

The advantages of a Fast-Flux scheme are as follows:

- **Simplicity**
  Maintaining content on one server is easier than maintaining it on all the bots.

- **Protection**
  The extra layers of redirection make it harder for researchers to find the server that actually hosts the content in a reverse proxy scheme.

- **Extended lifespan**
  If researchers do not fully reverse engineer the bot software it is hard to detect whether the bot uses Fast-Flux and where the content actually comes from. Researchers may spend time investigating machines that are actually disposable assets instead of focussing on the actual content hosting server.

Overview

Table 2.1 shows an overview of the three DNS based techniques described above. The "Speed" column in this table denotes the relative speed with which DNS records are changed. Regular use of DNS (e.g. no flux) in botnets means records are only changed in case of a takeover attempt, therefore the relative speed is "slow". All other techniques are "fast". The "Flux scope" column denotes which DNS parts of the botnet are fluxed, e.g. DNS entries in that scope are changed regularly.

2.2.4 Miscellaneous botnet enhancing techniques

We will now discuss some other botnet enhancing techniques.

Encoding, obfuscation and encryption

A lot of the research into botnets these days revolves around reverse engineering bot software binaries and network protocols[25][26][14][16]. Botmasters use a variety of techniques to hamper researchers in these areas.

To make the code of botnet software harder to understand, botmasters use encoding and obfuscation techniques. These techniques rearrange and reformat binary code, making it harder to understand the flow of the program and often making it hard to read the "useful" code without actually running the binary. "Packers" can be used to transform the binary into a self-extracting archive of sorts, where only the unpacking code is directly readable through disassembly and the packed code is not readable until after unpacking. Besides deterring reverse engineering efforts, these techniques can also be used to fool antivirus software based on malware

Name	Speed	Flux scope	Example botnet
Regular DNS use	Slow	None	Cutwail[23] & Grum[77]
Domain flux	Fast	Finding the C&C	Torpig[24] & Kraken[78]
Single Fast-Flux	Fast	Malicious content	Warezov/Stration[76]
Double Fast-Flux	Fast	DNS server and malicious content	Storm[79]

Table 2.1: *Botnet enhancing techniques using DNS. The Speed column denotes the relative speed with which DNS records are changed. The Flux scope column denotes which parts of the botnet using DNS are fluxed.*

signatures. Encoding and obfuscation methods raise the resilience of the botnet, as well as the stealth if they are used to evade detection by antivirus software.

Besides encoding and obfuscation, encryption techniques are also used to hide a bot's intentions. Encryption of bot communication[25] as well as encryption of the bot binary itself[14] has been seen in the wild. Both these forms of encryption raise resilience and stealth, as they mask information from researchers, making it hard to detect and understand the botnet.

TOR/VPN

Regardless of which control mechanism is used, in the end the botmaster will need to connect to her botnet command infrastructure to provide commands to her bots. Rather simple but effective measures can be taken in this phase to make sure that it is hard to track the origin and content of the connection, namely by using Tor[27] or a Virtual Private Network (VPN).

Tor is an anonymity providing proxy based on layered (or onion-like) encryption. It assures anonymity even in a situation where multiple nodes in the network have been compromised, meaning that it is extremely difficult for researchers to trace a connection back to its source. Although at the time of writing, no fully Tor based botnet designs have been found in the literature, there have been some proposals on what such a design should look like[80].

Besides using Tor a botmaster can easily set up a VPN service on the server running the command infrastructure (if there is such a server). Although a VPN server does not hide the connection's origin it will encrypt all traffic between the botmaster and the C&C, meaning that it is hard to perform network analysis on captured network traffic. Such a VPN has been used in the field, for instance in the Bredolab and Mariposa botnets[28].

Although TOR and VPN do not explicitly hide a botnet from outside spectators, they do provide extra anonymity for the botmaster, so both TOR and VPN usage raise botnet stealth.

Interfering programs

As more and more offenders see the benefits of using botnets to make money, competition between botmasters rises. Computers may get infected with different pieces of malware belonging to different botnets and botmasters may not be willing to share their profits with other botmasters. Recently, pieces of malware have begun to emerge with the ability to remove other bots from an infected machine, ensuring sole dominion over the infected system. An example of this is the SpyEye bot, which features a "bot-killer" for its most prominent rival, the ZeuS bot[29]. This bot-killer tries to find out whether the ZeuS malware is in-

stalled on the infected host and if it is, shut it down and remove it from the system.

Besides defences against other bots, botmasters have also built defences against antivirus software, as seen in recent Agobot instances[30]. This last feature makes botnets more resilient against removal from the infected system and also helps the bot stay hidden from the user, provided that the antivirus software is inactivated in such a way that it still seems active to the target.

2.2.5 Future botnet developments

As researchers progress their efforts in understanding, disrupting and taking down botnets, botmasters will need to resort on their ingenuity to come up with new defense mechanisms, new control channels and new ways of spreading their malware. In this section we'll take a brief look at what we can still expect in the future.

Different control channels

Other communication channels have been considered besides P2P, IRC and HTTP. The academic world has evaluated numerous possibilities, hoping that their research can preemptively identify novel botnet communication channels so that defensive measures can be designed before botmasters decide to use those channels. Examples of such new communication channels are Bluetooth[31], e-mail[32], social media (such as Twitter)[33] and even SMS[34]. Moving to these new channels means that the tried and true botnet detection and takedown procedures may no longer work. This could lead to delays in the judicial process or even to researchers failing to detect a botnet altogether.

Mobile botnets

Just like in the general virus world[35], botmasters seem to become interested in mobile phones as well. Starting with the SymbOS/Yxes[36] worm, serious advances have been made in building mobile malware capable of being controlled from the internet. One of the first fully-fledged mobile botnets was built around an iPhone ssh vulnerability, exploited by iKee.B[37]. This bot can pull new commands from a webserver using the phone's internet connection and is able to spread to other phones by exploiting the same vulnerability on phones connected to the local network. Because more and more people use their phones for mobile banking, mobile malware may the new platform of choice for botmasters[38]. So far there have been no reports of C&C infrastructures moving to mobile platforms as well.

Botnet builders

While building and running one's own botnet is a profitable business[81], it brings with some risks that not all botmasters are willing to take; getting caught is always a real possibility. Therefore, some choose to take botnet business exploitation to the next level, designing and building "bot kits" which customers can buy, after which customers can create their own botnet with just a few mouse clicks. Examples of this are the ZeuS and SpyEye kits.

Paying customers can buy additional modules if they need specific functionality such as an information stealer or a form grabber for a specific browser. They can also buy updated malware in case too many systems have had vulnerabilities patched and the existing malware starts losing potential. Lastly, if customers find issues within the kit or within the bots that it builds, the developer provides full support[82][83].

This commercial availability and easy setup make botnets accessible for a much broader group of offenders, leading to a proliferation of small botnets. We already see this happening with the ZeuS and SpyEye kits[3].

2.3 Botnet detection

The goal of much of the research into botnets is to provide support to law enforcement agencies and other parties in their efforts to disrupt or take down botnets. Besides research into the botnets themselves another big branch of research is geared towards developing methods and tools for detecting botnets. In this section we will take a brief look at the basics of botnet detection.

In our overview we will make a distinction between two areas of research: detection based on malicious traffic and detection based on C&C traffic. We will discuss both areas and give some examples of research.

2.3.1 Malicious traffic-based detection

The first area is focused on detecting botnets by monitoring network traffic and trying to find the malicious traffic that bots generate, i.e. DDoS, spam or malware infection traffic. Within this area a further distinction can be made between signature-based and anomaly-based techniques.

Signature-based techniques focus on deriving signatures of traffic or behaviour from known botnets. These signatures can then be used to search for similar traffic in a live network setting. Two examples of signature-based detection techniques are Snort[39] (an open source Intrusion Detection System) and Rishi[40] (a signature-based botnet detector for IRC-based botnets). A disadvantage of any signature-based detection method, be it for botnets or otherwise, is the reliance on knowing the signatures in advance; if no signatures are known for a botnet, a signature-based detection method will not be able to detect this botnet. This deficiency can be mitigated partially by generating the signatures automatically, such as proposed by Perdisci *et al.* in their work on clustering and signature generation for HTTP-based malware[41], although they still require a labeled malware training set.

This disadvantage also brings us to the second technique: anomaly-based detection. Instead of relying on patterns that need to be known in advance, anomaly-based detection focuses on network traffic anomalies, such as high traffic volumes, suspicious packet sizes or anomalies in network flow data. Early work in this area was performed by Binkley *et al.* [42] in 2006, who developed a detection methodology for IRC-based botnets based on the observation that *"IRC hosts are grouped into channels by a channel name [..] and that an evil channel is an IRC channel with a majority of hosts performing TCP SYN scanning"*[42]. They analysed a year's worth of network traffic to create 2 sets of groups of hosts. One set consists of groups of hosts that are in the same IRC channel based on IRC commands in the captured TCP packets. The second set consists of all hosts that have a high "work load", defined as the ratio of TCP control packets versus the total number of TCP packets. This metric is meant to find hosts that are performing port scan activities as their work load is higher than the work load of normal hosts. If a high percentage of hosts in an IRC channel also has a high work load, it means they are probably all performing port scans and chances of those hosts being part of a botnet are high.

A bit later, in 2007, Gu *et al.* [43] designed and implemented BotHunter, a bot-

[3]See the ZeuS and SpyEye trackers at `http://abuse.ch`

net detection system based on following the dialog between a host trying to infect another host. They modelled the infection process as a loosely ordered series of information exchanges, such as port scanning, inbound exploitation, new malware downloading and C&C communications. They combine this model with information from an IDS in which they model each of these information exchanges using packet contents, netflow data and payload signatures for known botnet. By correlating the information from the IDS with the infection model they are able to detect bot infections in the monitored network.

The same team also designed BotMiner in 2008[44], which incorporates both malicious traffic and Command & Control traffic. They look at two types of traffic, designated by them as C-plane ("Who is talking to whom") and A-plane ("Who is doing what") traffic. The C-plane traffic is captured by a regular netflow capture tool. This "C-flow" data shows which hosts were talking to each other during a day. The flows are clustered by projecting them into a space and then using Ξ-means clustering. The A-plane traffic is captured by a modified version of BotHunter. Each record in the A-plane is grouped by recorded activity and then each group is clustered further using specific activity features. For instance, for the activity "sending spam" one could cluster using the destination of the SMTP connection or the content of the spam message. After both the C-plane an A-plane data is clustered, a cross-plane correlation is performed to determine which hosts performed similar activities in the A-plane *and* appear in the same C cluster. This finally shows which hosts are probably part of the same botnet.

Work by Ehrlich *et al.* [45] is a methodology geared towards spam. Using a classifier based on a Bayesian classification of a labeled training set, they divide spam and non-spam SMTP traffic. They then determine whether the spam senders also communicate with a C&C by looking at the entropy of the local and remote ports used for each connection and comparing that entropy to models of known botnet spam senders and known non-botnet spam senders. They then try to find this C&C by combining several flow-based metrics with DNS metadata to find a single source of control for all the found spam sources, making this another method combining malicious traffic and C&C traffic.

Each of the detection methods has its own shortcomings. Lying a focus on IRC botnets only, such as Binkley *et al.* have done, may no longer be sufficient, as botnets are moving away from their IRC base. The BotHunter detection process relies, in part, on the detection of inbound exploitation attempts, by using Snort rules for specific exploits. In the validation they provide for their method, almost every successful detection was based on detecting a known botnet malware. It is unclear whether their methods work as well as they claim for unknown pieces of malware. Furthermore, their detection method is based on a bot communication model, something that may change for future botnets. BotMiner incorporates BotHunter as part of its detection method, making it vulnerable to the same weaknesses. None of these detection methods are capable of detecting the C&C if they find a botnet, they require additional analysis of the malware. The spam-based method proposed by Ehrlich *et al.* uses traffic statistics from SMTP servers to determine whether a bot sends spam or not. Botmasters can evade detection by using legitimate mail accounts, using stolen webmail login credentials.

It is also interesting to note that the detection methods mentioned above focus on specific malicious activities: malware infections, port scanning and sending spam. None use malicious activities such as DDoS, information stealing or click fraud; it may be prudent to develop detection methods based on these activities.

2.3.2 Command & Control traffic-based detection

As the goal of botnet detection is usually to find the C&C and shut it down, focusing on malicious traffic might not be the best approach; instead one could also try to find the C&C directly, by finding and analyzing the communication between the C&C and the bots.

An interesting field of anomaly-based botnet detection research in this area is the research based on "IP flows", as we have already seen above. This relatively new field acknowledges the problem of using full packet inspection to analyse network traffic due to the sheer amount of processing required and instead focuses on aggregate information in the form of IP flows. These flows are gathered for every connection made through a central point (such as a router) from one host to another and encompass general network statistics about that connection, such as timestamps, the number of transferred bytes and the number of sent packets. Based on this information one can perform an anomaly-based traffic analysis to find indicators of botnet traffic[46].

In 2007, Karasaridis *et al.* [47] provided a detection method for IRC-based botnets based on IP flows. They propose a multistage approach starting with trigger information of suspicious activities from distributed consumer IDS systems to find suspicious hosts. All flows from and to these hosts are gathered and filtered using three different approaches. Firstly, all traffic to known IRC ports is gathered. Secondly, if a host has many incoming connections it may be a control hub, so traffic to such a hub is gathered as well. Lastly, they used an (undocumented) flow model for IRC traffic to classify flows that are probably IRC traffic. This filtered data, which they call "Candidate Controller Conversations", is then filtered again in two steps. First, if many flows point towards the same destination address and destination port, that destination is marked as a possible C&C and all flow sources are marked as suspected bots. All destinations are rated on the number of suspected bots and destinations which fall below a certain threshold are discarded for further evaluation. All flows to the remaining destination hosts are compared to the IRC traffic model again, to determine which destination address/port combinations are most interesting. For each item in this final list of address/port combinations a heuristics score is determined, based on how periodic the flows are for the item. All these calculated metrics are combined in a confidence score, which finally rates how likely it is that a given host is a C&C.

Later, in 2008, Gu *et al.* [48] designed BotSniffer. BotSniffer uses IP flow information along with payload information to determine which nodes in a network respond to the same query within the same time frame and with a high similarity in payload content. Using this *"spatial-temporal correlation and similarity"* they can find which computers are part of a botnet.

A different approach was taken by Livadas *et al.* [49], who use machine learning techniques to first distinguish IRC from non-IRC traffic and then to distinguish botnet IRC traffic from non-botnet IRC traffic. They evaluate different machine learning methodologies (such as J48, naive Bayes and Bayesian network classifiers) and conclude that, although difficult and quite sensitive to the quality of the training data, using machine learning for botnet detection should be feasible.

Later work by Strayer *et al.* [68] and Ehrlich *et al.* (see above) proves this conclusion by using various machine learning techniques to detect botnet C&C traffic and ultimately, the C&C itself. The work by Strayer *et al.* uses a set of increasingly more complex analyzers to find out which hosts are part of a bot-

net and determine the botnet C&C. They start with some basic filtering of the flow data based on white/blacklists and some simple flow attributes to remove non-TCP traffic, port scans, high-bitrate flows, flows with very large packets and flows with a short duration. Then a classifier tries to determine which flows have chat-like characteristics, using machine learning classification algorithms. This step was unsuccessful in their training set as the false positive and false negative rate was too high for all tried algorithms, but can probably be improved with a better training set. The third step is a flow correlation step, which is implemented in a fashion similar to the C-flow clustering mentioned earlier in the BotMiner summary. Each flow is projected in a d-dimensional space, using different attributes of the flow for each dimension, and the normalized Euclidean distance between two flows determines whether flows are close to each other; if the distances are close to 0 chances are good that the flows are somehow related. The final step in the determination process is to analyze the clusters of flows with a low distance to each other and look for the host that has the most incoming connections; this host is most likely the C&C, the other hosts are most likely bots.

Ramachandran *et al.* [50] use data on DNS blacklist lookups, which botmasters that want to send spam perform to see if their bots are placed in a DNS blacklist, as a measure to see which systems are infected. They build a DNS blacklist query graph from data gathered from the blacklist query logs from ISPs. An edge from A to B in such a graph denotes that host A queried the blacklist to see if host B is on it. They then examine the graph to look for two properties: Spatial relationships and Temporal relationships. A legitimate SMTP server will send queries to the blacklist to find out if connecting hosts are legitimate, but the server itself will be the subject of queries by other hosts as well, i.e. it is the object and subject of queries. A botmaster only queries about its own hosts, but is never the subject of queries, so this should be visible in the graph by nodes with a high out-degree and a low in-degree. For the second property, a comparison between the arrival of e-mail and the arrival of blacklist queries is performed. Queries from legitimate SMTP servers will correlate with the arrival of e-mail whereas queries from botmasters will not. This property wasn't tested by the authors and remains the topic of future work.

A final mention here should also go to BotMiner (see above), as their approach uses IP flow information to find hosts with similar traffic characteristics (i.e. bots).

As already discussed by Sperotto *et al.* [46], a lot of the C&C traffic based botnet detection methods are based on analysis of netflow data. Active evasion by botmasters could impair these detection methods. As they are all based on detecting anomalies in the traffic, botmasters could try to shape their bot traffic even more like regular HTTP traffic, effectively hiding their botnet. Another reason for concern is that none of these detection methods can detect P2P botnets. The dynamicity of the botnet world means that detection methods will have to evolve as well, if we are to stay ahead of the threat.

2.3.3 Comparison

In this section we will compare the mentioned detection techniques using some of the comparison metrics Feily *et al.* [51] used in their botnet detection survey. The results are presented in table 2.2. The "Low false positive" metric is high or low compared to the best false positive rates found in the field, especially those of BotMiner and the project by Ehrlich *et al*, which were both around 2%. In some cases no false positive and/or false negative measurements were performed or the measurements were too unclear for interpretation, in which case the table lists a question

Class	Ref	Name or researcher	Unknown bot detection	Protocol & structure independent	Encrypted bot detection	Low false positive
Malicious traffic based	[39]	Snort	×	×	✓	×
	[40]	Rishi	×	×	×	×
	[41]	Perdisci	×	×	×	✓
	[42]	Binkley	✓	×	✓	?
Hybrid	[43]	BotHunter	×	×	×	✓
	[45]	Ehrlich	✓	✓	✓	✓
	[44]	BotMiner	✓	✓	✓	✓
C&C traffic based	[47]	Karasaridis	✓	×	✓	✓
	[48]	BotSniffer	✓	×	✓	✓
	[49]	Livadas	✓	×	✓	×
	[68]	Strayer	✓	×	✓	?
	[50]	Ramachandran	✓	✓	✓	?

Table 2.2: *Comparison of botnet detection techniques*

mark. The "Encrypted bot detection" metric indicates whether a detection technique can detect bots even though the traffic that is analysed is encrypted.

It is worth noting that the low false positive rates are mainly encountered in later work (e.g. after 2008), which is to be expected in a starting field of research. Another observation is that there aren't many detection techniques that are protocol & structure independent. Given the variety of today's botnets and the ingenuity of the botmasters, it would be prudent to focus on developing more generally applicable detection methods in the future. A last observation is that the newest and most successful research uses information both from the malicious traffic as well as the C&C traffic.

2.3.4 Conclusion

In the past pages we have discussed botnets and botnet detection methods. We have looked at the general design, at botnet uses, at botnet characteristics and at the ways researchers try to detect botnets.

We see that botnets grow in sophistication over time, as do the detection methods provided by the literature. Recent news items[84][85][86] show the interest in botnets by law enforcement agencies and large software companies alike.

As already discussed in the introduction, all detection methods we have seen focus on network traffic in one form or another. In the next few chapters we will discuss a novel botnet detection method based on file system indicators. We will show the results of applying our method to a test dataset and discuss some of the advantages and shortcomings.

Chapter 3

Method

We set ourselves the goal of determining whether a given computer system contains a botnet C&C or not. As discussed in the introduction we want to focus on the file system, so we assume that we only have access to the computer's hard disk. In the forensics world it is customary to process hard disk information in the form of disk images, created by tools such as "dd", "FTK" or "EnCase", so we assume that the disk information from the computer(s) we want to research comes in the form of a disk image. This disk image can be created from a physical or a virtual machine.

There are multiple ways of looking at hard disk information. One way is to process the information available through the file system. This means that the file system needs to be intact, uncorrupted and unencrypted. Using this method one can look at the information present on the disk as if it were a regular local disk, including directories, files and metadata. Another way of looking at hard disk information is by ignoring the file system altogether and processing the raw data stream available from a hard disk. An advantage of this method is that all information on the disk will be processed, even information that is hidden outside the regular file system boundaries. The first method may miss information if the file system is corrupt, if file systems are removed from a disk or if a suspect hides files outside the file system on purpose. A disadvantage of the second method is that every single byte of the disk needs to be processed, even if most of the disk contains garbage data, making this method slow compared to the first method. It also has another problem: if we can't use information from the file system we do not know where files start and end on the disk, meaning that finding the contents of a single file may be cumbersome.

For our botnet detection, we don't need to have the guarantee that all possible evidence is found and processed. Furthermore, we try to identify simple indicators which can be located relatively fast, so we will use the first of the methods described above. In this context the word fast means that we don't want to have to parse every byte of every file on the file system, looking for indicator matches.

3.1 Dataset

Before we dive into the indicators themselves, we need to discuss the dataset we use to derive the indicators from, as well as the dataset that we use to test our detection method. The dataset that we use consists of a total of 207 disk images that were created from 129 confiscated computers in the "Tolling" case, run by the National High Tech Crime Unit of the Dutch National Crime Squad. The total size of the dataset is 23TB of compressed EnCase images, about 57TB of uncompressed data. The images have been created using the EnCase® Forensic software.

The Tolling case was a botnet takedown case in 2010, in which the C&C for the Bredolab botnet was taken down. Strictly speaking there is no such thing as *the* Bredolab botnet, as the malware that was used seems to keep resurfacing under different names. The name Bredolab is most often used to denote this specific botnet setup, however, so we will use it in this fashion.

The C&C infrastructure itself consisted of 6 different computers, each with its own function. Besides the Bredolab computers a lot of other computers were confiscated in the same case, all of which belonged to the same bulletproof hosting provider. A bulletproof hosting provider is an individual or organisation that provides webhosting or dedicated servers with a high level of leniency towards the content that is distributed with the provided webhosting or server. Bulletproof hosting is used extensively in the spamming and copyright infringing content distribution industries. In this particular case a bulletproof hosting provider called John Datri rented servers from LeaseWeb, one of the largest hosting providers in Europe. He then subleased these servers to third parties, with the guarantee that these third parties would remain anonymous with regards to LeaseWeb. The Bredolab botmaster rented his servers from this bulletproof hosting provider. There was sufficient evidence to suggest that the other computers that John Datri rented out were also involved in illegitimate businesses, be it botnet related or otherwise, so the District Attorney decided that these other computers should be confiscated as well.

For our research, we discard all images that have no valid file system. We can see whether a disk has a valid file system by looking at the Master Boot Record (MBR), to find out whether the disk is partitioned and if so, which file system type is used for each partition. To determine this information programmatically we use the open source forensics toolkit "The Sleuth Kit"[87]. 45 images were removed from the set in this step.

We also discard all images containing the Windows operating system, which removes another 60 images. The reason for this is that a lot of well-known botnet C&C's are known to run on Linux, including ZeuS, SpyEye, Flame[88] and Bredolab. Because of this, the indicators we have gathered come only from C&C's that are installed on Linux. The directory structures for Unix operating systems and the Windows operating system differ substantially, leading to a difference in indicators. Furthermore, the tools used most often for hosting websites and databases are different between Windows and Unix operating systems, meaning that searching for certain indicators may become difficult or even impossible. Adding support for searching Windows file systems as well remains future work.

After this filtering, 102 images remain, belonging to 90 different computers. Six of these images belong to the Bredolab C&C infrastructure. In order to validate our method we labelled each of the images in this final 102 as "botnet" or "clean", by manually looking through the file system on each image to determine whether it contains C&C related material. The distribution of these labels can be seen in table 3.1. All the Bredolab images are classified as "botnet". Within the rest of the images we have found that 18 images contain botnet control software, making a total of 24 botnet images.

3.2 Indicators

To determine whether a given system contains a botnet C&C or not, we want to search its file system for indicators that reveal the presence of a C&C. However, before we can start searching, we need to know which indicators to look for.

It is worth explaining our use of the term "C&C" for this section. There are two meanings to this term: the broad term for "entities that control the bots in botnets" and the narrow term used when indicating a specific C&C infrastructure for a specific botnet, i.e. "The Bredolab C&C". We will try to find indicators that are present in as many different C&C's as possible, in the narrow sense of the word. The distinction between these two meanings leads to two different approaches:

	Bredolab C&C	Other	Total
Botnet	6	18	24
Clean	0	78	78
No valid file system	0	45	45
Windows	0	60	60
Total	6	201	207

Table 3.1: *The number of Bredolab and other disk images*

One approach would be to build a database of the files making up the C&C's belonging to as many known botnets as possible, then look for those files on the file system, i.e. a signature-based approach. If a number of files in this database are found on the file system, we can be pretty sure that there is a C&C present. This method is similar to the way most antivirus programs try to detect viruses. Unfortunately, this would mean that we can only detect C&C's for known botnets, as our indicators would be too botnet specific.

Another approach would be to collect a number of C&C's for known botnets and try to distill generic C&C indicators from this collection, incorporating information from the literature and opinions of field experts. There is no guarantee that the indicators that result from this process will be present in the C&C's of future botnets, but it would be a step in the right direction. We have chosen to use this approach for our research. In the next few sections we will discuss how we came to our indicators.

We have used two sources for gathering known C&C's. The first source is the leaked versions of ZeuS and SpyEye, as discussed in section 1. For the second source we used part of our dataset. To this end, we divided the dataset into a "indicator group" and a "test group". In the indicator group, we included the full Bredolab infrastructure (6 images) and two other images labeled as botnet. All the other images are part of the test group. The full distribution of images can be seen in table 3.2.

Besides these two sources we have also spoken with experts in the field of botnet research and botnet takedowns. These are:

1. The digital investigators who were involved with the Bredolab case at NHTCU, who know the Bredolab dataset well and helped devise the indicators "File Type" and "Database table name".

2. The Director of the Digital Crimes Unit at Microsoft; Kayne Naughton at Shadowserver; a legal consultant at Fox-IT and a number of technical representatives of the NCSC (former GOVCERT), all of whom helped to validate the detection method.

Besides these expert opinions a number of academic publications have helped our understanding of the inner workings of certain common botnets, such as ZeuS[15], SpyEye[29] and Cutwail[23]. These papers show how more and more bots use the http protocol to communicate with their C&C, which in turn tells us something about the way the C&C is set up.

Below follows a list of the indicators we have gathered. For each indicator we discuss the reasoning behind it and we will say something about the robustness of the indicator. The robustness of an indicator shows how hard it is for a botmaster to adapt their C&C software such that the indicator no longer applies. For instance, the robustness of the "Database table name" indicator is "fragile", because a botmaster can change the names of the tables the C&C software uses, after which we can

	Indicator group	Test group	Total
Botnet	8	16	24
Clean	0	78	78
Total	8	94	102

Table 3.2: *The number of disk images in the indicator and test groups*

no longer detect the C&C using that specific indicator.

3.2.1 Database table name

The Bredolab C&C, as well as the ZeuS and SpyEye C&C all use suspicious names for the tables in the database that they use to store information about each bot. Examples are "bm2bots_stats" or "4bnevbotstatus". We try to find out where the database is located on disk by parsing the configuration files for the database software. We then extract the database table names and see if they contain the word "bot".

Robustness: fragile, because a botmaster can change the names of the tables used by the C&C software.

3.2.2 File size

Law enforcement see full disk encryption and encrypted vaults more and more often in their cases[89][90][91]. It makes sense for a botmaster to put the vital parts of the C&C in an encrypted vault as it will make forensics of the machine virtually impossible without getting hold of the encryption key or accessing the machine while the vault is already opened. If the botmaster ever gets caught by law enforcement, hiding the C&C software and data in an encrypted vault may deny law enforcement access to crucial evidence to make a case.

However, simply looking for large files on the file system gives us many false positives. If the server is used for sharing movies or hosting virtual machines, for instance, large files will be present, but the machine doesn't necessarily contain an encrypted vault. Calculating the entropy of each large file also doesn't solve this problem as movies or other large files may have the same entropy characteristics as an encrypted vault and on top of that, calculating the entropy of a file takes a lot of time.

We try to lower the number of false positives by marking an image as suspicious if and only if the number of files over 1GB is larger than 0 and less than 5. Of course, even if the large file is an encrypted vault, this does not necessarily mean that the machine is botnet related. The next chapter will show whether this indicator yields many false positives or false negatives.

Robustness: medium, if a botmaster wants to use encrypted vaults these will usually be detectable. A botmaster can, however, use full disk encryption instead of encrypted vaults, in which case there is no access to the file system and thus no files to look for.

3.2.3 SQL queries

This indicator is an extension of the database table name indicator mentioned above. For all web hosting related scripts we find on a file system, we try to see if hard coded SQL queries contain the word "bot" using a regular expression and if so, mark the system containing the scripts as suspicious.

Robustness: fragile, as with Table names, a botmaster can change the SQL queries to circumvent the indicator.

3.2.4 File type

Bredolab was a so-called pay-per-install botnet. The botmaster used Bredolab as a platform to distribute other pieces of malware for third parties in exchange for money. As we only look at Unix operating systems the presence of Windows binaries, especially if they're known pieces of malware, is suspicious. We use the libmagic library[1] to discover the file type of files in the file system. If a file is a Windows binary, we calculate its SHA-1 hash. We use this hash to search through the Virustotal[2] and Team Cymru[3] malware hash databases to see if the file is a known piece of malware.

Robustness: medium, a botnet needs (Windows) binaries to spread, so these will usually be present. The indicator can be fooled, though, by storing the binaries in encrypted format, or in a compressed file.

3.2.5 Spam lists

The final indicator in our list has everything to do with spam sending botnets. Many botnets are being used to send spam[92]. Sending spam requires access to large amounts of e-mail addresses. The two extra images we have used in our training set contain plain text files with many e-mail addresses, sometimes multiple gigabytes. We search the files on the file system for e-mail addresses using a regular expression and if a file contains more than 500 e-mail addresses we mark the system containing the file as suspicious. The reason we have set this number is because the spam lists we have encountered are usually quite large. They contain several million e-mail addresses, making processing each of these files completely an arduous task, so we limit the amount of processing performed.

Robustness: medium, spam sending botnets will always need access to large numbers of e-mail addresses, but a botmaster can try to hide the spam lists, for instance by encoding or encrypting them.

3.2.6 Web script file name

In section 2 we have already seen that many botnets are controlled via the HTTP protocol. We also see this in the Bredolab, ZeuS and SpyEye C&C's. We can use the indicator above (database table name) for web hosting related scripts as well: some of the PHP scripts in the SpyEye C&C also have "bot" in their filename. Therefore, we look for files related to web hosting (PHP files, HTML files, CGI files, etc.) with the word "bot" in their filename.

Robustness: fragile, because the botmaster can change the names of the web scripts the C&C software comprises of.

3.2.7 Web script entropy

Another interesting characteristic of the Bredolab C&C files is that the web hosting related scripts used to control and monitor the bots were all encoded with the Zend Guard[4] encoder. Zend Guard can be used to encode and obfuscate PHP files, to make reading or changing the original source harder for researchers or customers of the botnet. We have not seen this behaviour in any other C&C in our indicator group, but it makes sense for developers of crimeware kits to want to hide their source files to protect their business model; if anybody could modify and resell the C&C source they would soon be out of business.

[1] `http://linux.die.net/man/3/libmagic`
[2] `https://www.virustotal.com/`
[3] `https://www.team-cymru.org/Services/MHR/`
[4] `http://www.zend.com/en/products/guard/`

We try to find such encoded web script files by looking at their entropy. The entropy of a file shows how "random" the content a file is. If we compare the entropy of the Bredolab PHP files with the entropy of the PHP files in some well known PHP frameworks (Wordpress, Zend, CodeIgniter and CakePHP) we see that the entropy of the Bredolab PHP files is much higher than the entropy of the PHP files in the frameworks. The encoding used for the Bredolab files ensures that the content has a high entropy, a high randomness. The files in the PHP frameworks are not encoded or obfuscated, so their randomness is a lot lower and thus so is their entropy. The Shannon[93] entropy scale ranges from 1 to 8, where 8 denotes the highest entropy and 1 is the lowest. The Shannon entropy number describes how many bits would be required to encode all the symbols of a given file: the higher the number, the more bits are required, meaning that more different symbols are present in the file, which means a higher entropy. The Bredolab files all have an entropy between 7 and 8, whereas the "regular" files have an entropy below 6. Encoding and obfuscation techniques work on any web hosting related script, so we measure the entropy for every web hosting related script and look for scripts that have a Shannon entropy above 6.

Unfortunately, encoding and obfuscation techniques are used in legitimate businesses as well, by software development companies that want to keep the source of their products hidden. Therefore, the false positive ratio for this indicator may be quite high, depending on how many of these legitimate uses we come across.

Robustness: fragile, if a botmaster wants to hide his code he doesn't have much choice but to use obfuscation/encoding, but botnets such as ZeuS and SpyEye do not use encoding/encryption like the Bredolab botnet does.

3.2.8 Other indicators

There were some other ideas for indicators that never made it off the drawing board. These are:

Log parsing

One could parse log files for web servers or other services to see if many incoming or outgoing connections were seen to many different IP addresses in a short time. These communications could indicate that the system was sending commands to or receiving content from bots. However, differences between the way these log files are formatted make this a difficult task, so we did not implement this indicator.

Known botnet files

One could also gather as many different C&C files as possible, then search for the presence of these files on the system. Beside the reasons already mentioned in the beginning of this section, gathering these different C&C files is a difficult task. Laws in different countries, organizational policies and different jurisdictions often make it difficult to get access to the files, even if they are readily available for employees.

Popular exploitation tools

Botmasters often use exploitation kits, such as the Blackhole exploit kit[94], to infect their targets. They may also hack web servers to use as a distribution platform for the botnet malware. It is reasonable to assume that they have a number of popular hacking tools available on their systems. One could build an indicator to search for these tools. There are a number of reasons we did not implement this indicator. First of all, we did not find any exploitation related software other than

the actual botnet malware on the machines in the indicator set. Secondly, even if a botmaster used popular hacking tools they would most likely be present on his desktop, not on his C&C machine. Lastly, building a database of the files of such hacking tools would most likely require more time than we have available for this research.

3.3 Detection methodology

Now that we have defined the indicators we look for, we will discuss the exact method of searching for these indicators.

3.3.1 Disk images

As already mentioned above, the test group we have defined consists of 94 EnCase images. Each of these images contains a (compressed) byte-by-byte copy of the disks of the confiscated systems. To get access to the raw file system, we first use the "xmount"[5] tool to expose the uncompressed data contained in the EnCase image. This process creates a virtual file representing the raw data of the disk image on the local computer. We then use The Sleuth Kit to identify which file system type is used and where each partition resides on the image, measured as an offset in bytes. With this information we can mount each partition locally, read-only, using the default Linux application "mount", giving us access to the files and directories on each partition. We then proceed with processing this file system tree, searching for the indicators mentioned above.

3.3.2 The search process

We search by recursively walking over every directory in the file system. For each directory we look at which files are available. A file can be either a regular file or a "special" file, such as a "symbolic link" or a "block special file". As our indicators focus on file metadata and content, we are only interested in regular files. The other types of files are used by the system for a variety of purposes but explaining these is outside the scope of this paper. We match the file with each of the indicators, to see if one or more indicators have a hit.

File filtering

Some of the indicators suffer from a high false positive ratio due to the presence of a specific set of files; the same files are present on a lot of different images. We can identify and filter these files before any of the indicators are tested. The filtering is performed by comparing the name of each file to a list of patterns and when a pattern matches, remove the file from further consideration for this image. The following files or filename patterns are filtered out:

- **changelog**
- **/var/lib/rpm/packages**
- **.cpan/metadata**
- **/var/lib/apt/lists**

These are files that are often present in software source code distributions, containing information on how to contact the programmers that work on a piece of software. This leads to a large number of e-mail addresses when a lot of different programmers are involved, which in turn leads to a false positive for the Spam List indicator for every file.

- **/mail/root**

The pattern /mail/root is found in the files `/var/spool/mail/root` and `/var/mail/root`. Both these files contain mail used by the Linux system to relay logging information to the root user. This internal mail is addressed to an e-mail address,

[5] `https://www.pinguin.lu/index.php`

leading to false positives for the Spam List indicator.

- **/var/log/btmp**

The `/var/log/btmp` file logs each failed login attempt, both for local and remote (ssh) access. If a server has had a public function, chances are that ssh brute force scripts have tried to log into the server using commonly used user/password combinations. As these logins will fail for any properly managed server, they are logged into the `/var/log/btmp` file. Because of this, the file will grow in size quickly, generating false positives for our Big Files indicator.

- **robotparser.py**
- **robotparser.html**
- **bottom.html**
- **bottom.php**

Both the `robotparser` files are part of the standard library of the Python programming language, meant to assist in parsing `robots.txt` files. The two `bottom` files are found in numerous legitimate web frameworks, such as Squirrelmail, and the Zend framework. These four files cause false positives for the Web script name indicator; they contain the word "bot" even though they are not botnet related.

- **/doc/cups**

Files contained in a `/doc/cups` directory are packaged with the CUPS printing system. They are documentation files, usually in HTML, available in different languages, amongst which are Korean and Japanese. As these languages use characters that are not available in the regular ASCII character set, these files are encoded using UTF-8, which uses two bytes for each single character. On top of that, these languages have more characters than the Latin alphabet. These two facts combined cause the entropy of these files to rise above the threshold, which in turn causes the Web script entropy indicator to generate false positives.

3.3.3 Proof of concept implementation

The proof of concept implementation we have built is written in the Python programming language. It works in two stages. In the first stage it performs all of the steps discussed above automatically for a given list of disk images. After the mounting process it walks through the file system in search of files. It feeds the location of every file it encounters to a list of plugins, each of which searches for one or more indicators.

If an indicator matches (e.g. a file is found containing a lot of e-mail addresses) a report is generated containing the following details:

- The name of the image
- The full path to the file within the image that matched the indicator
- The indicator that matched
- Some relevant details of the matched indicator and file

The details included in this report depend on the indicator. For instance, if a file is matched by the Spam lists indicator we include the file type and if it is matched by the File size indicator we include the file size.

In the second phase, it performs some post-processing on the generated report. The system that has access to the disk images may not at the same time have access to internet, due to the fact that the images are considered case evidence. This impairs our File type indicator, as we cannot see if the found Windows binaries are known pieces of malware. In this second phase we remove any File type indicator hit that is not in one of both databases.

We will try to determine which indicators are best at correctly identifying images that have

a botnet C&C. For each indicator, we will classify an image as suspicious for that indicator if one or more files are found on the image that match the indicator. We will determine false positive and false negative ratio's by comparing each classification to the labels we have given each image; we say that an image is classified correctly by an indicator if there was an indicator hit for a file on the image and the image is labeled as "botnet". If an indicator classifies an image as suspicious, but the image does in fact not contain a C&C we say that the indicator generates a false positive for that specific image. If, on the other hand, an image does contain a C&C but an indicator does not have any hits for the image, we say that the indicator generates a false negative for that specific image. A correct positive classification is called a true positive; a correct negative classification is called a true negative. We calculate the false positive ratio by using the following formula:

$$FPR = \frac{FP}{FP + TN}$$

We calculate the false negative ratio by using the following formula:

$$FNR = \frac{FN}{FN + TP}$$

In the next chapter we will discuss the results of applying our proof of concept implementation to our test dataset. We will also discuss some advantages and disadvantages of our method and of each chosen indicator.

Chapter 4

Results

As we have seen in the previous section, the dataset that we have used in our experiments consists of 94 disk images, with a total of 18 verified botnet C&C's.

4.1 Results per indicator

We have calculated the false positive and false negative ratio's for each of the indicators. These results can be found in table 4.1.

We can see that some indicators perform better than others. The Database table name indicator and SQL queries indicator perform best on average, both with a low FPR and average ratio. We also see that the Spam lists indicator has a high FPR of 33.33%, meaning that an image that does not contain a C&C has over 30% chance of being classified wrong by this indicator. The File type indicator has a low FPR and an FNR of almost 47%, so it misses a large number of C&C's, even though it almost never misclassifies any clean machines.

The question is which of the two ratio's has more impact on the practical applications of file system based botnet classification: FPR or FNR. A high FPR means that when the classification is used in a triage procedure there will be a lot of machines that are falsely accused of containing a C&C. A high FNR means that some machines that do contain a C&C will be missed. In this use case we will probably favour a high FPR over a high FNR: missing a C&C machine is probably worse than the extra work involved in investigating a machine that turns out to be a clean.

As is usually the case in a classification procedure, indicators with a low FPR have a high FNR and vice versa; there is no indicator with both a low FPR and a low FNR. Also, the indicators with a low average error ratio (below 20%) are all fragile; they can be circumvented easily.

4.2 Combining indicators

One way to solve this problem is by finding better, more robust indicators, but we leave this endeavor for future work. Another way would be to "smooth" the rules we use to define when an image is suspicious. Instead of using a single indicator we can try to combine indicators; we say that an image is suspicious when a predefined number of indicators match, regardless of which ones match.

Table 4.2 shows the FPR and FNR results for this experiment. The headers of the columns denote the number of indicators that should match before we regard an image as suspicious. We have added a robustness estimate to each column. The robustness lowers as more indicators are needed for a positive match: the chance that a botmaster circumvents every indicator is much lower than that a botmaster circumvents one or two indicators.

As expected, when only a single indicator is needed for positive classification the FPR is very high and the FNR is very low; the chance that an indicator matches, regardless which one, is quite high. We see that as more indicators are needed for a positive classification, the FPR lowers and the FNR rises. Figure 4.1 shows a visualisation of this. This figure also

	Database table name	File size	SQL queries	File type	Spam lists	Web script file name	Web script entropy
TOTAL	94	94	94	94	94	94	94
FP's	0	18	2	4	39	19	19
FN's	6	11	5	14	2	5	4
FPR	0%	19%	3%	5%	33%	20%	20%
FNR	27%	41%	24%	47%	11%	24%	20%
Average	14%	30%	13%	26%	22%	22%	20%
Robustness	Fragile	Medium	Fragile	Medium	Medium	Fragile	Fragile

Table 4.1: *FPRs per indicator*

	1	2	3	4	5	6	7
TOTAL	94	94	94	94	94	94	94
FP's	43	34	18	6	0	0	0
FN's	2	2	3	5	7	13	15
FPR	36%	30%	19%	7%	0%	0%	0%
FNR	11%	11%	16%	24%	30%	45%	48%
Average	23%	21%	17%	15%	15%	22%	24%
Robustness	Robust	Robust	Medium	Medium	Medium	Fragile	Fragile

Table 4.2: *FPRs when using multiple indicators*

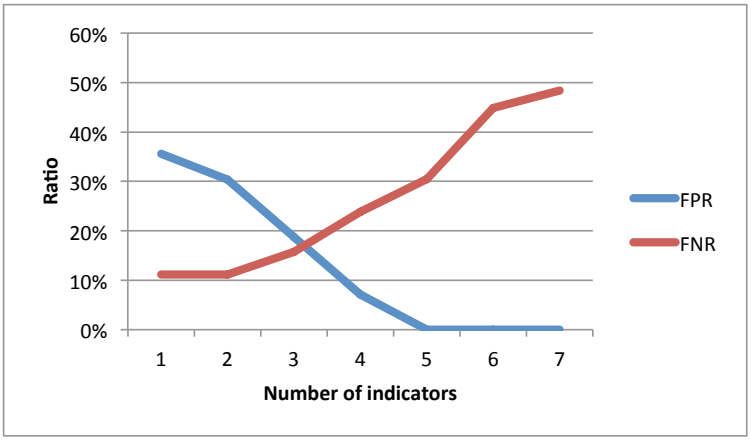

Figure 4.1: *Graph of FPR and FNR for different numbers of indicators*

shows where the equal error ratio (EER) resides: between 3 and 4 required indicators.

It is interesting to note that we can reach practically the same false positive and false negative ratio's for this combined indicator method as we could for the method using individual indicators. We can see this when we use 4 or 5 indicators: the average FPR and FNR ratio's are comparable to the best individual indicators, between 13 and 15 percent.

4.3 Indicator performance analysis

The indicators Database table name and SQL queries perform better than the other indicators, especially regarding false positives. An explanation for this is that both metrics revolve around the same principle: database tables with names containing the word "bot". So, either many C&C databases have this particular trait or all the C&C's present in our test dataset belong to the same botnet family. The latter explanation is viable as we have already seen that the crimeware kit business flourishes, based largely around the ZeuS and SpyEye kits, which both use database tables with "bot" in the name.

There are some parts of our indicators that could be improved:

- The File size indicator does find a number of encrypted containers, but some of the C&C machines in the test set don't use encrypted vaults. Other large files that it finds are usually logging files or spam lists.
- The File type indicator usually gets it right when it matches, but many of the C&C's in the test apparently store the malware on a different machine than the one we have in our possession.
- As we have already seen in the section on filtering certain files, the Spam lists indicator matches on any file containing a lot of e-mail addresses, including a lot of legitimate files. Lowering the FPR ratio for this indicator even further may be accomplished by filtering out more files, by raising the number of required e-mail addresses or possibly by tweaking the indicator implementation to better match the specific characteristics of spam lists such as e-mail host heterogeneity and storage format.
- The Web script file name and Web script entropy indicators share the same issue: many files share the same characteristics as the C&C related files, causing the indicator to match too often.

Combining indicators by requiring a number of them to match before declaring an image as suspicious helps to raise the overall accuracy. All in all we can see that the number of machines that an investigator will have to analyze manually can be reduced by as much as much as 60% while keeping the number of false negatives low. Further research may take this number even higher.

Chapter 5

Conclusion

In the past pages we have discussed botnets and botnet detection methods. We have looked at the general design, at botnet uses, at botnet characteristics and at the ways researchers try to detect botnets.

We have proposed a novel botnet detection method using file system indicators. We have seen that we can use individual indicators to detect botnets with a low false positive rate and that using a combination of indicators raises the robustness of the detection method, while keeping the false positive and false negative ratio's within bounds.

5.1 Future work

Unfortunately, there are also a number of shortcomings to our proposed method. We will discuss these shortcomings in the following sections, providing ideas on how to solve the shortcoming in each section.

First of all, none of the proposed indicators have both a low false positive and a low false negative rate. Using a combination of indicators does not solve this problem, at least not in the simple manner in which we combine them. One could think of more intricate ways of combining the indicators, weighing each indicator's strengths and weaknesses.

Secondly, none of the indicators we have found have a high robustness; each indicator could be circumvented by the botmaster, albeit with some work. Developing more robust indicators by further studying the file system characteristics of C&C's would help solve this problem.

A third shortcoming is that our method is based solely on Unix like operating systems. The first Windows based C&C has already been seen in the Mariposa botnet[25], which would not have been detected by our method. Adapting our detection method to provide support for the Windows file system and developing indicators that work on Windows as well as on Linux is required to detect as many C&C's as possible.

The dataset we have used to test our detection method comes from a single source and may therefore not be the best representation of the real world. To further validate our methods, they should be applied to various other data sources, preferably from parties other than law enforcement, to make sure they work well in all scenario's.

A final mention should go to the speed and efficiency of the proposed method. The entire classification process for every image in our test set took little over 93 hours. Of this total time, 53 hours were spent on only 2 images, both of which contain a large number of small and similar files: session files for a PHP website. Anomalies such as this can be expected when searching through large numbers of images, each of which was owned by a different person. Enhancing the detection method to deal with such anomalies in a smart way could lower the running time significantly.

We see that botnets grow in sophistication over time, as do the detection methods provided by the literature. Recent news items[84][85][86] show the interest in botnets by law enforcement agencies and large software companies alike. But as the battle rages on, it is only a matter of time before botmas-

ters become so proficient in hiding their botnets that fighting them becomes very much a cat-and-mouse-game where the mice are numerous, invisible and more vicious than the cat.

Appendix A

Proof of concept implementation

A.1 Modules

The proof of concept implementation consists of the following modules. The number after each module indicates the number of lines of source code, excluding comments, for the module (SLOC).

classify.py (62)

This is the main module which parses the list of disk images that the user provides, handles the logging setup and makes sure reports are generated based on the results of each plugin.

image_parser.py (160)

This module mounts each EnCase image, searches through the image to look for partitions using The SleuthKit python bindings and mounts each partition. It then walks over every file available in the file system on the partition and calls each plugin for each file.

util.py (120)

This module contains a number of utility functions for file path manipulation, hashing and functions used to call external libraries such as xmount and mount.

[1] http://docs.python.org/2/c-api/

plugins/1_table_names.py (63)

This plugin handles the indicator Database table name, by parsing database configuration files to look for the location on the file system where the data files for the database are stored. It then searches through these files to look for suspicious table names.

plugins/2_suspicious.py (129)

This plugin handles all other indicators. The reason all these indicators are handled by a single plugin is because indicators may reuse information already generated by other indicators for the same file, e.g. if an indicator matches and generates a SHA-1 hash, a later indicator does not need to recalculate this hash. Another reason is that we only want to load and process the raw data of a file once, instead of loading the contents of a file once for every indicator.

plugins/entropy/entropy.c (49)

Calculating the Shannon entropy of a file requires us to look at every byte of raw data of a file and perform a number of calculations for every byte. Instead of performing this process in pure Python code we created a Python API[1] based module written in C, which proved to be a factor 20 faster than the same implementation written in pure Python. Because this module uses the Python API it can be loaded as a Python module for easy access.

tools/lookup_hashes.py (56)
tools/virustotal.py (24)
tools/team_cymru.py (12)

These three modules are used in the second phase of the C&C searching process, where the results for the indicator File type are filtered to include only known malware binaries.

A.2 Performance

A total of 23 TB of compressed EnCase images takes little over 93 hours to process using our proof of concept implementation. In comparison: the manual classification we performed to produce our labeled test dataset took about 2 weeks. Speed improvements could be made, however, in the following areas:

A.2.1 Different programming language

We chose Python as the programming language for our proof of concept implementation because of it's simplicity and it's availability of different libraries. Using a "faster" programming language such as Java or even C(++) may improve speed.

A.2.2 Parallel processing

Our proof of concept implementation handles every image in a serial fashion. While processing an image, one CPU core of our test system was fully saturated, whereas the network link was not. Because we handle each image separately several of them could be processed in parallel so we make use of as many CPU cores as possible, until the network link is saturated.

A.2.3 Smart file selection

As already mentioned in the conclusion, more than half of the total running time was spent only two images containing millions of small empty files. These files provide a challenge because a request has to be made to access each file, even though the file is empty. Devising a smarter way of selecting which files to process could decrease the running time.

Glossary

botmaster An individual or group in control of a botnet. 5

botnet A large group of networked zombie computers under centralised control. 3

bulletproof hosting provider An individual or group providing access to anonymous and abuse tolerant web hosting or dedicated servers. 3

C&C An infrastructure used to control the bots in a botnet, either centralised or decentralised. 3

disk image A byte-by-byte copy of the hard disk of a computer. 4

EnCase EnCase is a company that creates software to perform forensics on systems, including EnCase Forensic which can create raw byte-by-byte copies of hard disks and perform analysis on the resulting disk image. 22

file system A means to organize data on a hard disk, usually in the form of directories containing files. 4

netflow Abstract captured network data including observed connections and high level statistics. 3

takedown Destroying or disrupting botnet activities. 3

triage Deciding which element, from a set of elements, requires immediate attention. 3

Peer-reviewed literature

[1] D. Dagon, G. Gu, C. P. Lee, and W. Lee, "A Taxonomy of Botnet Structures," in *Proceedings of the 23rd Annual Computer Security Applications Conference ACSAC '07*, vol. 36, (Miami Beach, Florida), pp. 325–339, IEEE, 2007.

[2] H. R. Zeidanloo and A. A. Manaf, "Botnet Command and Control Mechanisms," in *Second International Conference on Computer and Electrical Engineering ICCEE '09*, (Dubai), pp. 564–568, IEEE, 2009.

[3] D. Dittrich and S. Dietrich, "Command and control structures in malware: From Handler/Agent to P2P," *USENIX ;login:*, vol. 32, no. 6, pp. 18–27, 2007.

[4] B. B. Kang, D. Dagon, Y. Kim, E. Chan-Tin, C. P. Lee, J. Tyra, H. J. Kang, C. Nunnery, Z. Wadler, G. Sinclair, and N. Hopper, "Towards complete node enumeration in a peer-to-peer botnet," in *Proceedings of the 4th International Symposium on Information Computer and Communications Security ASIACCS '09*, (Sydney), pp. 23–24, ACM, 2009.

[5] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "My botnet is bigger than yours (maybe, better than yours): Why size estimates remain challenging," in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets, HotBots ' 07*, (Cambridge, MA), pp. 1–8, USENIX Association, 2007.

[6] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis, "A multifaceted approach to understanding the botnet phenomenon," in *Proceedings of the 6th ACM SIGCOMM on Internet measurement IMC '06*, (Rio de Janeiro), pp. 41–52, ACM Press, 2006.

[7] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov, "Spamming Botnets : Signatures and Characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 171–182, 2008.

[8] J. Nazario, "DDoS attack evolution," *Network Security*, vol. 2008, no. 7, pp. 7–10, 2008.

[9] J. Caballero, C. Grier, C. Kreibich, V. Paxson, and U. C. Berkeley, "Measuring Pay-per-Install : The Commoditization of Malware Distribution," in *Proceedings of the 20th USENIX conference on Security SEC '11*, (San Francisco, CA), pp. 13–13, USENIX Association, 2011.

[10] M. Gandhi, M. Jakobsson, and J. Ratkiewicz, "Badvertisements: Stealthy Click-Fraud with Unwitting Accessories," *Journal of Digital Forensic Practice*, vol. 1, no. 2, pp. 131–142, 2006.

[11] J. P. Farwell and R. Rohozinski, "Stuxnet and the Future of Cyber War," *Survival: Global Politics and Strategy*, vol. 53, no. 1, pp. 23–40, 2011.

[12] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A Survey of Botnet Technology and Defenses," in *Conference for Homeland Security, 2009 CATCH '09*, (Washington, DC), pp. 299–304, IEEE, 2009.

[13] C. Li, W. Jiang, and X. Zou, "Botnet: Survey and Case Study," in *Proceedings of the 4th International Conference on Innovative Computing Information and Control ICICIC '09*, (Kaohsiung), pp. 1184–1187, IEEE, 2009.

[14] K. Chiang and L. Lloyd, "A Case Study of the Rustock Rootkit and Spam Bot," in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets, HotBots ' 07*, (Cambridge, MA), pp. 1–10, USENIX Association, USENIX Association, 2007.

[15] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang, "On the Analysis of the Zeus Botnet Crimeware Toolkit," in *Eighth Annual International Conference on Privacy, Security and Trust PST '10*, (Ottowa, ON), pp. 31–38, IEEE, 2010.

[16] N. Daswani and M. Stoppelman, "The Anatomy of Clickbot.A," in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets, HotBots ' 07*, (Cambridge, MA), pp. 1–11, USENIX Association, 2007.

[17] I. Arce and E. Levy, "An analysis of the Slapper worm," *IEEE Security & Privacy*, vol. 1, no. 1, pp. 82–87, 2003.

[18] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich, "Analysis of the Storm and Nugache Trojans: P2P is here," *USENIX ;login:*, vol. 32, no. 6, pp. 18–27, 2007.

[19] D. Dittrich and S. Dietrich, "P2P as botnet command and control: A deeper insight," in *3rd International Conference on Malicious and Unwanted Software, 2008 MALWARE '08*, (Fairfax, VI), pp. 41–48, IEEE, 2008.

[20] P. R. Marupally and V. Paruchuri, "Comparative Analysis and Evaluation of Botnet Command and Control Models," in *24th IEEE International Conference on Advanced Information Networking and Applications AINA '10*, (Perth, WA), pp. 82–89, IEEE, 2010.

[21] R. Link and D. Sancho, "Lessons learned while sinkholing botnets - Not as easy as it looks!," in *Proceedings of the 21st Virus Bulletin International Conference*, (Barcelona), pp. 106–110, Trendmicro, 2011.

[22] J. Nazario and T. Holz, "As the net churns: Fast-flux botnet observations," in *Proceedings of the 3rd International Conference on Malicious and Unwanted Software MALWARE '08*, (Fairfax, VI), pp. 24–31, IEEE, IEEE, 2008.

[23] B. Stone-gross, T. Holz, G. Stringhini, and G. Vigna, "The Underground Economy of Spam: A Botmaster's Perspective of Coordinating Large-Scale Spam Campaigns," in *Proceedings of the 4rd USENIX conference on Largescale exploits and emergent threats botnets spyware worms and more*, (Boston, MA), pp. 4–4, USENIX Association, 2011.

[24] B. Stone-gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna, "Your Botnet is My Botnet : Analysis of a Botnet Takeover," *Security*, vol. 97, no. 3, pp. 635–647, 2009.

[25] P. Sinha, A. Boukhtouta, V. H. Belarde, and M. Debbabi, "Insights from the Analysis of the Mariposa Botnet," in *Fifth International Conference on Risks and Security of Internet and Systems CRiSIS '10*, (Montreal, Canada), pp. 1–9, IEEE, 2010.

[26] G. Sinclair, C. Nunnery, and B. B. H. Kang, "The waledac protocol: The how and why," in *4th International Conference on Malicious and Unwanted Software MALWARE '09*, (Montreal, Canada), pp. 69–77, IEEE, 2009.

[27] R. Dingledine, N. Mathewson, and P. Syverson, "Tor: The Second-Generation Onion Router," in *Proceedings of the 13th USENIX Security Symposium*, (San Diego, CA), pp. 303–320, USENIX, USENIX Association, 2004.

[28] D. Dittrich, "So You Want to Take Over a Botnet ...," in *Proceedings of the 5th USENIX Workshop on Large-Scale Exploits and Emergent Threats LEET '12*, (San Jose, CA), pp. 1–8, USENIX Association, 2012.

[29] A. Sood, R. Enbody, and R. Bansal, "Dissecting SpyEye - Understanding the Design of Third Generation Botnets," *Computer Networks*, 2012.

[30] J. Liu, Y. Xiao, K. Ghaboosi, H. Deng, and J. Zhang, "Botnet: Classification, Attacks, Detection, Tracing, and Preventive Measures," *EURASIP Journal on Wireless Communications and Networking*, vol. 2009, no. 1, pp. 1–12, 2009.

[31] K. Singh, S. Sangal, N. Jain, P. Traynor, and W. Lee, "Evaluating Bluetooth as a Medium for Botnet Command and Control," in *Proceedings of the 7th international conference on Detection of intrusions and malware, and vulnerability assessment DIMVA '10*, vol. 2010, (Bonn), pp. 61–80, Springer Verlag, 2010.

[32] K. Singh, A. Srivastava, J. Giffin, and W. Lee, "Evaluating email's feasibility for botnet command and control," in *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC DSN '08*, (Anchorage), pp. 376–385, IEEE, 2008.

[33] E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu, "Social Network-Based Botnet Command-and-Control: Emerging Threats and Countermeasures," in *Proceedings of the 8th international conference on Applied cryptography and network security ACNS '10*, (Beijing), pp. 511–528, Springer, 2010.

[34] G. Geng, G. Xu, M. Zhang, Y. Guo, G. Yang, and C. Wei, "The Design of SMS Based Heterogeneous Mobile Botnet," *Journal of Computers*, vol. 7, no. 1, pp. 235–243, 2012.

[35] M. Hypponen, "Malware goes mobile.," *Scientific American*, vol. 295, no. 5, pp. 70–77, 2006.

[36] A. Apvrille, "Symbian worm Yxes: Towards mobile botnets?," *Journal in Computer Virology*, vol. 8, pp. 1–24, 2012.

[37] P. Porras, H. Saidi, and V. Yegneswaran, "An Analysis of the iKee.B iPhone Botnet," *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, vol. 47, no. 5, pp. 141–152, 2010.

[38] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, "A survey of mobile malware in the wild," in *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices SPSM '11*, vol. 55, (Chicago, IL), pp. 3–14, University of California, ACM Press, 2011.

[39] M. Roesch, "Snort - Lightweight Intrusion Detection for Networks," in *Proceedings of the 13th conference on Systems Administration LISA '99*, (Seattle, WA), pp. 229–238, USENIX Association, 1999.

[40] J. Goebel and T. Holz, "Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation," in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets, HotBots ' 07*, (Cambridge, MA), pp. 1–8, USENIX Association, 2007.

[41] 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*, (San Jose, CA), p. 14, USENIX Association Berkeley, CA, USA 2010, 2010.

[42] 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 SRUTI'06*, (San Jose, CA), p. 7, USENIX Association, 2006.

[43] G. Gu, P. Porras, V. Yegneswaran, M. Fong, W. Lee, and M. Park, "BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation," in *Proceedings of 16th USENIX Security Symposium SS'07*, (Boston, MA), p. 16, USENIX Association, 2007.

[44] 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 USENIX Security Symposium SS'08*, (San Jose, CA), pp. 139–154, USENIX Association, USENIX Association, 2008.

[45] W. K. Ehrlich, A. Karasaridis, D. Liu, and D. Hoeflin, "Detection of spam hosts and spam bots using network flow traffic modeling," in *Proceedings of the 3th USENIX Workshop on Large-Scale Exploits and Emergent Threats LEET "10*, (San Jose, CA), p. 7, USENIX Association, USENIX Association, 2010.

[46] A. Sperotto, G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller, "An Overview of IP Flow-Based Intrusion Detection," *IEEE Communications Surveys Tutorials*, vol. 12, no. 3, pp. 343–356, 2010.

[47] A. Karasaridis, B. Rexroad, and D. Hoeflin, "Wide-scale Botnet Detection and Characterization," in *Proceedings of the First Workshop on Hot Topics in Understanding Botnets, HotBots ' 07*, vol. 07, (Cambridge, MA), pp. 1–7, USENIX Association, 2007.

[48] G. Gu, J. Zhang, and W. Lee, "BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic," in *Proceedings of the 16h Annual Network & Distriuted System Security Symposium NDSS'08*, (San Diego, CA), pp. 1–13, Citeseer, 2008.

[49] C. Livadas, R. Walsh, D. Lapsley, and W. Strayer, "Using Machine Learning Techniques to Identify Botnet Traffic," in *Proceedings of the 31st IEEE Conference on Local Computer Networks LCN'06*, (Tampa, FL), pp. 967–974, IEEE, 2006.

[50] A. Ramachandran, N. Feamster, and D. Dagon, "Revealing botnet membership using DNSBL counter-intelligence," in *Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet SRUTI'06*, vol. 36, (San Jose, CA), p. 8, USENIX Association, 2006.

[51] M. Feily, A. Shahrestani, and S. Ramadass, "A Survey of Botnet and Botnet Detection," in *Proceedings of the 3rd International Conference on Emerging Security Information Systems and Technologies SECURWARE '09*, (Athens), pp. 268–273, IEEE, 2009.

Other literature

[52] D. Fisher, "Zeus Source Code Leaked." https://threatpost.com/en_us/blogs/zeus-source-code-leaked-051011, 2011.

[53] S. Bodmer, "First Zeus, now SpyEye. Look at the source code now!." https://blog.damballa.com/archives/1357, 2011.

[54] N. Anderson, "Massive DDoS attacks target Estonia; Russia accused." http://arstechnica.com/security/2007/05/massive-ddos-attacks-target-estonia-russia-accused/, 2007.

[55] D. E. Sanger, "Obama Order Sped Up Wave of Cyberattacks Against Iran." http://www.nytimes.com/2012/06/01/world/middleeast/obama-ordered-wave-of-cyberattacks-against-iran.html?_r=2&pagewanted=2&seid=auto&smid=tw-nytimespolitics&pagewanted=all, 2012.

[56] A. Kadiev, "End of the Line for the Bredolab Botnet?." http://www.securelist.com/en/analysis/204792152/End_of_the_Line_for_the_Bredolab_Botnet, 2010.

[57] N. Ianelli and A. Hackworth, "Botnets as a Vehicle for Online Crime," Tech. Rep. 1, CERT/CC, 2005.

[58] Trend Micro, "Cashing in on Cybercrime - New Malware Target Bitcoin," tech. rep., Trend Micro, Cupertino, CA, 2012.

[59] FireEye, "Advanced Threat Report - 2H 2011," tech. rep., FireEye, 2012.

[60] S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," Tech. Rep. 3, bitcoin.org, 2009.

[61] P. Quinn, "Report: Bitcoin. Open, decentralized and growing.." http://p-quinn.com/entries/report/report-bitcoin-open-decentralized-and-growing-, 2012.

[62] G. Ollmann, "The Opt-In Botnet Generation," tech. rep., Damballa, 2010.

[63] Q. Norton, "Anonymous 101 Part Deux: Morals Triumph Over Lulz." http://www.wired.com/threatlevel/2011/12/anonymous-101-part-deux/3/, 2011.

[64] S. Golovanov and I. Soumenkov, "TDL4 Top Bot." http://www.securelist.com/en/analysis/204792180/TDL4_Top_Bot, 2011.

[65] R. Mullins, "The biggest cloud on the planet is owned by ... the crooks." http://www.networkworld.com/community/node/58829, 2010.

[66] R. Pointer, "Eggdrop." http://www.eggheads.org, 1998.

[67] B. Barlowe, J. Blackbird, W. S. Davis, and Others, "The evolution of malware and the threat landscape - a 10-year review," tech. rep., Microsoft, 2012.

[68] W. T. Strayer, D. Lapsely, R. Walsh, and C. Livadas, "Botnet Detection Based on Network Behavior," in *Botnet Detection* (W. Lee, C. Wang, and D. Dagon, eds.), no. August in Advances in Information Security, ch. 1, pp. 1–24, Washington, DC: Springer, 2008.

[69] Team Cymru and S. Santorelli, "Episode 77: Dead Botnets." http://www.youtube.com/watch?v=vYTM9s15yk4, 2010.

[70] J. Zhuge, T. Holz, and X. Han, "Characterizing the IRC-based Botnet Phenomenon," tech. rep., Peking University & University of Mannheim, 2007.

[71] P. Barford and V. Yegneswaran, "An Inside Look at Botnets," in *Malware Detection (Advances in Information Security)* (M. Christodorescu, S. Jha, D. Maughan, D. Song, and C. Wang, eds.), Advances in Information Security, ch. 8, pp. 171–191, Springer Verlag, 2006.

[72] G. Tenebro, "The Bredolab Files." http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the_bredolab_files.pdf, 2009.

[73] N. Fitzgibbon and M. Wood, "Conficker.C A Technical Analysis," tech. rep., SophosLabs, Sophos Inc., 2009.

[74] J. Stewart, "Protocols and Encryption of The Storm Botnet." https://media.blackhat.com/bh-usa-08/video/bh-us-08-Stewart/black-hat-usa-08-stewart-stormbotnet-hires.m4v, 2008.

[75] G. Ollmann, "Botnet Communication Topologies," tech. rep., Damballa Inc., Atlanta, 2009.

[76] William Salusky and R. Danford, "Know Your Enemy: Fast-Flux Service Networks," *The Honeynet Project Website*, vol. 114, no. 4, pp. 1–24, 2007.

[77] A. Mushtaq, "Killing the Beast - Part 5." http://blog.fireeye.com/research/2012/07/killing-the-beast-part-5.html, 2012.

[78] P. Royal, "Analysis of the Kraken Botnet," tech. rep., Damballa, 2008.

[79] P. Porras, H. Saidi, and V. Yegneswaran, "A Multi-perspective Analysis of the Storm (Peacomm) Worm," Tech. Rep. 650, Computer Science Laboratory, SRI International, 2007.

[80] D. Brown, "Resilient Botnet Command and Control with Tor." http://www.defcon.org/images/defcon-18/dc-18-presentations/D.Brown/DEFCON-18-Brown-TorCnC.pdf, 2010.

[81] Z. Li, Q. Liao, and A. Striegel, "Botnet Economics: Uncertainty Matters," in *Managing Information Risk and the Economics of Security* (E. Johnson, ed.), pp. 245–267, Hanover: Springer, 2009.

[82] K. Stevens and D. Jackson, "ZeuS Banking Trojan Report." http://www.secureworks.com/research/threats/zeus/, 2010.

[83] A. K. Sood, R. J. Enbody, and R. Bansal, "SpyEye malware infection framework." http://www.virusbtn.com/virusbulletin/archive/2011/07/vb201107-SpyEye.dkb, 2011.

[84] Microsoft, "Microsoft Joins Financial Services Industry to Disrupt Massive Zeus Cybercrime Operation That Fuels Worldwide Fraud and Identity Theft." http://www.microsoft.com/en-us/news/press/2012/mar12/03-25CybercrimePR.aspx, 2012.

[85] FBI, "Operation Ghost Click." http://www.fbi.gov/news/stories/2011/november/malware_110911/malware_110911, 2011.

[86] D. Fisher, "Microsoft, FireEye Take Down Notorious Rustock Botnet." http://threatpost.com/en_us/blogs/microsoft-fireeye-take-down-notorious-rustock-botnet-031811, 2011.

[87] B. Carrier, "The Sleuth Kit." http://www.sleuthkit.org/sleuthkit/, 2012.

[88] "GReAT", "Full Analysis of Flame's Command & Control servers." http://www.securelist.com/en/blog/750/Full_Analysis_of_Flame_s_Command_Control_servers, 2012.

[89] D. McCullagh, "Feds seek new ways to bypass encryption." http://news.cnet.com/8301-31921_3-20035168-281.html, 2011.

[90] D. McCullagh, "FBI to announce new Net-wiretapping push." http://news.cnet.com/8301-31921_3-20032518-281.html, 2011.

[91] H. S. N. Wire, "Feds forced to get creative to bypass encryption." http://www.homelandsecuritynewswire.com/feds-forced-get-creative-bypass-encryption, 2011.

[92] J. Orloff, "Top Spam Botnets Over Time." http://www.allspammedup.com/2011/03/top-spam-botnets-over-time/, 2011.

[93] C. E. Shannon and W. Weaver, "Mathematical Theory of Communication." http://www.amazon.com/Mathematical-Theory-Communication-Claude-Shannon/dp/0252725484, 1949.

[94] F. Howard, "Exploring the Blackhole exploit kit." http://nakedsecurity.sophos.com/exploring-the-blackhole-exploit-kit/, 2012.