GENERATING CONTENT-BASED SIGNATURES FOR DETECTING BOT-INFECTED MACHINES

A THESIS
SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Leyla Bilge
July, 2008

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Ali Aydın SELÇUK(Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. İbrahim KÖRPEOĞLU

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Mustafa AKGÜL

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. BARAY
Director of the Institute

ABSTRACT

GENERATING CONTENT-BASED SIGNATURES FOR DETECTING BOT-INFECTED MACHINES

Leyla Bilge
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. Ali Aydın SELÇUK
July, 2008

A botnet is a network of compromised machines that are remotely controlled and commanded by an attacker, who is often called the botmaster. Such botnets are often abused as platforms to launch distributed denial of service attacks, send spam mails or perform identity theft. In recent years, the basic motivations for malicious activity have shifted from *script kiddie* vandalism in the hacker community, to more organized attacks and intrusions for financial gain. This shift explains the reason for the rise of botnets that have capabilities to perform more sophisticated malicious activities. Recently, researchers have tried to develop botnet detection mechanisms. The botnet detection mechanisms proposed to date have serious limitations, since they either can handle only certain types of botnets or focus on only specific botnet attributes, such as the spreading mechanism, the attack mechanism, etc., in order to constitute their detection models.

We present a system that monitors network traffic to identify bot-infected hosts. Our goal is to develop a more general detection model that identifies single infected machines without relying on the bot propagation vector. To this end, we leverage the insight that all of the bots get a command and perform an action as a response, since the command and response behavior is the unique characteristic that distinguishes the bots from other malware. Thus, we examine the network traffic generated by bots to locate command and response behaviors. Afterwards, we generate signatures from the similar commands that are followed by similar bot responses *without* any explicit knowledge about the command and control protocol. The signatures are deployed to an IDS that monitors the network traffic of a university. Finally, the experiments showed that our system is capable of detecting bot-infected machines with a low false positive rate.

Keywords: botnet, botmaster, malware.

ÖZET

BOTLAR TARAFINDAN ELE GEÇİRİLMİŞ BİLGİSAYARLARIN TESPİT EDİLMESİ İÇİN İÇERİK-TABANLI İMZALARIN ÜRETİLMESİ

Leyla Bilge
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Assist. Prof. Dr. Ali Aydın SELÇUK
Temmuz, 2008

Botnet'ler botmaster adı verilen saldırganlar tarafından uzaktan kontrol edilip yönetilebilen, ele geçirilmiş makinalardan oluşan ağlardır. Botnetler genelde dağıtık hizmet engelleme saldırıları uygulamak, reklam içerikli e-posta göndermek ya da kimlik hırsızlığı yapmak için kullanılırlar. Son yıllarda, kötü niyetli faliyetlerdeki temel amaç, haker topluluğundaki özenti çocukların saygınlık kazanma isteklerinden daha çok organize saldırılarla finansal kazanç sağlamaktır. Bu değişim, daha sofistike kötü niyetli faliyetleri yapabilme özelliği olan botnetlerin sayısındakı artışın nedenini de açıklar. Son zamanlarda, araştırmacılar botnetleri yakalamak için yoğun çalışmalar yapmaktalar. Şimdiye kadar geliştirilen sistemler, bazı bot özelliklerine, çoğalma yöntemlerine ya da saldırı şekillerine odaklandıkları için ne yazık ki çok sınırlıdırlar.

Biz, ağ trafiğini izleyerek, yerel ağdaki bot tarafından ele geçirilmiş makinaları tespit eden bir sistem sunuyoruz. Bizim amacımız, bot yayılma vektöründen bağımsız bir şekilde ele geçirilmiş makinaları tespit eden daha genel bir yakalama yöntemi geliştirmektir. Bunun için, botların en belirgin karakteristiği olan komut alma ve komuta itaat etmek özelliğinden yararlanıyoruz. Bot tarafından üretilmiş ağ trafiğini inceleyip, komutları ve cevaplarını tespit ediyoruz. Ardından, belirli bot davranışlarını tetikleyen benzer komutlardan, komut ve kontrol protokolü hakkında bir ön bilgiye sahip olmadan bot yakalama imzaları üretiyoruz. Ürettiğimiz imzalar, bir üniversitenin trafiğini izleyen ve denetleyen bir IDS'e uygulanmıştır. Yaptığımız deneylerin sonunda, bizim sistemimizin bot tarafından ele geçirilmiş makinaları çok düşük orandaki yanlış alarmlar ile yakaladığı ortaya çıkmıştır.

Anahtar sözcükler: botnet, botmaster, malware.

Acknowledgement

This masters thesis was carried out in cooperation with Peter Wurzinger at International Secure Systems Lab in Technical University of Vienna. I am glad that I had the possiblity to work with Peter on a very interesting and hot subject. I would like to thank him for being a great project partner.

Moreover, I would like to thank my advisors Engin Kirda and Christopher Krügel for their patience and support during the whole work.

I also would like to thank to my advisor Ali Aydın Selçuk in Bilkent University that he gave me the chance to be an exchange student in Technical University of Vienna.

Special thanks to my office mates Manuel Egele, Martin Szydlowski and Clemens Kolbitsch for their friendship and sharing the tea break with me.

My final thanks are for Tübitak which supported me financially during my master studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

During the last ten years, the increase in the popularity of the Internet caused people to be addicted to this phenomenon to such an extent that it is almost impossible for most of the users to complete any piece of work without the help of the Internet. This major growth in popularity results in the increase of the number of *cyber-criminals* as well. The miscreants do not need to expend any physical effort to steal money from banks as it so was in the past. Internet-based attacks are relatively easy to launch. Moreover, it is difficult to chase and catch the criminals, since all of the activities on the Internet are done virtually. Thus, the Internet became an attractive tool for the attackers who are inclined to use it for their nefarious purposes.

The most dangerous, effective and popular tool of choice for cyber-criminals today are *bots* [19]. A Bot (a.k.a. *zombie* or *drone*) is a compromised machine that can be controlled by an attacker remotely. Immediately after the bot binary is installed on the target machine either by exploiting known vulnerabilities or by using social engineering techniques, a command and control channel (C&C channel) is established between the bot and the controller, who is called as the *botmaster*. One of the most distinguishing characteristics of bots [6] is to establish a C&C channel, which allows the botmaster to remotely control or update the compromised machine. In order to perform more effective attacks, the *botmaster* tries to compromise several machines to construct a *botnet*, a network of bots.

Such botnets are often abused as platforms to launch distributed denial of service attacks, to send spam mails or to steal secret information.

Traditional means of defense against malware can be either host-based or network-based. Today, the most preferable host-based defense systems are anti-virus systems that periodically scan the computer to detect malware. The defense provided by the anti-virus programs relies on pre-defined signatures that are supposed to identify the malware. Unfortunately, such malware detection schemes have limited capabilities [4] against bots that have a fast evolution which is difficult to be kept up by the anti-virus programs, since the pre-defined signatures they made use of are generated with a manual effort. To mitigate this limitation, another type of host-base defense systems, which use static [5, 18] or dynamic [15, 38] code analysis techniques to extract the behavior of unknown programs, have been proposed. Although these systems are able to identify the malicious behavior correctly, because of their run-time overhead, they are not reasonable enough to be applied to detect bots.

Contrary to host-based analysis techniques, the network-based analysis techniques require the users of the computers to install neither an anti-virus program nor an analysis platform. Typically, they deploy an intrusion detection system (IDS) to monitor network traffic for signs that indicate the presence of the malware. Clearly, the same technique can be applied to detect bot-infected machines in a network. The first system that tries to detect bot-infected machines by network traffic analysis was BotHunter [10], a system that correlates three different IDS alerts to identify bots. Unfortunately, the bot propagation model the authors present is quite limited, since most of the stages that characterize a bot infection consist of scanning activity and remote exploits. Therefore, the system cannot detect bots that do not propagate by exploiting vulnerabilities, but using social engineering techniques such as deceiving the user to open an attachment or click a link on a web page.

Another recent system that analyzes network traffic to find signs for bot infection is BotSniffer [11]. BotSniffer correlate the network activity of machines, which are in the same network, to detect members of the same botnet. Since the members of a botnet take the same command simultaneously, their response to the command will be same as well. Thus, if there are enough amount of bots that are members of the same botnet, the system is able to correlate the command and response activity performed by the bots and botmaster to detect the insider bots. Of course, the system has limited capabilities to detect individual bot-infected machines. Because system requires observing at least two bots that behave same in order to correlate a network activity.

We present a system that monitors network traffic to identify bot-infected hosts too. However, our goal is to develop a more general detection model that identifies single infected machines without depending on the bot propagation vector. The unique characteristic that distinguishes the bots from other malware is that they can be remotely controlled by a botmaster. When the botmaster wants to start an activity, such as scanning activity, denial of service attack, etc., she simply sends the command to the bots and expects that the bots obey to the command by carrying out some actions. In order to generate bot detection models, we leverage the fact that all of the bots get a command and perform an action as a response. Thus, we examine the network traffic to locate command and response behaviors. Afterwards, we generate signatures from the commands that are followed by bot responses. The signatures generated have the appropriate format that can directly be deployed to popular IDSs, such as Bro [24] and Snort [28]. Since our analysis focuses only on the command and response activity, our system can detect bots completely independently from their spreading vector. Also, we can detect bot-infected machines, even if there is only one in the network, because the IDSs are capable of detecting malware without concerning the number of infected machines.

There is a growing variety of different bot families which have different command sets and of course, different corresponding bot responses. Therefore, we analyze different bot families individually and generate specific detection models that are applicable only for the bots that are members of a specific bot family. To this end, we cluster bot binaries in a way that the bot binaries belong to the same bot family are grouped together. Once the bots are grouped, we insert the data collected for each family as an input to our system.

Another salient feature of our system is that we can automatically generate signatures by observing the network traffic generated by real bots that are captured in the wild. The real traffic is collected by executing each bot binary in a controlled environment and recording its network activity. To this end, while a bot is run in our test environment, we do not restrict the network accessibility of the bot, but allow it to establish the connection to the botmaster. Since there is no restriction, the bot can also perform the malicious activity that is commanded by the botmaster. Therefore, we can observe all of the commands and responses from the network captures.

Chapter 2

Botnets

2.1 Definition of Bots and Botnets

Basically, bot (also know as *zombie* or *drone*) is a compromised machine that can be controlled by an attacker remotely. The bot binary might be installed on the target machine either by exploiting a known vulnerability or by using social engineering techniques such as deluding the Internet user to click a link that might be send by an e-mail or MSN Messenger chat. As soon as the bot binary is executed, it connects to the *botmaster* in order to get commands. The ability to be remotely controlled and commanded is the most important property of bots and this ability also distinguishes them from other malwares.

Botnet is a network that consists of several malicious bots that are controlled by a commander. Typically, the bot controller, which is actually called as *botmaster*, uses a command and control channel (C&C channel) in order to send her commands that demand some malevolent activities to be performed. The botnets have a reputation on the influential distributed denial of service attacks. Since the more bots in the botnet, the more powerful attacks could be performed; the botnets need a propagation mechanism to increase their population. Generally, they make use of some off-the-shelf propagation mechanisms that are also used by existing worms.

2.1.1 Historical Evolution of Botnets

Bots, which are one of the most dangerous malwares nowadays, interestingly, were invented for benign usage. The first bots were programs that worked in IRC [13] network. In late 1980s, the IRC platform was developed for providing a chatting service to several users, and bots were used to entertain users by offering them game or message services. After a while, the attackers found a way to abuse bot usage and waged *IRCwars*. The *IRCwars* were one of the first documented distributed denial of service attacks.

In late 1999, SANS Institute researchers discovered remotely executable code on thousands of Windows machines. They were inspired by remote control nature of the code while they were naming the infected computers as *robots*, which is shortened to *bot* later. Because the code was encrypted, the researchers could not easily reverse-engineer it to determine what the purpose of the code was until the bots did one-week-long distributed denial of service attack that targeted Amazon, eBay and other secure ecommerce sites in February 2000.

The Table 2.1 provides a timeline that ranges from the first popular *IRC* bot *EggDrop* to recently released peer-to-peer bot *Storm*. First malicious bots used Microsoft IRC client, *mIRC.exe*, with slight modifications for commanding the bots. Then, more modular, robust and effective bots that had their own IRC clients were developed. Malicious bots have seen much development in the recent years after the emergence of peer-to-peer bots. Some peer-to-peer bots used existing protocols while the others developed new protocols to construct their networks.

Now, botnets of more than a million compromised computers are found regularly in the wild, although they usually run in packs of 10 to 20,000 to avoid detection. They have a very big ration in 50 top malware list of well-know malware analyzer companies. Thus, they can be referred as one of the most powerful threats against Internet users.

Date	Name	Description
12/1993	EggDrop	non-malicious IRC bot
04/1998	GTbot	Malicious IRC bot based on MIRC
04/2002	SDbot	Provided own IRC Client
10/2002	Agobot	Robust, flexible, modular design
04/2003	Spybot	Extensive feature set based on Agobot
03/2004	Phatbot	P2P bot based on WASTE
03/2006	SpamThru	P2P bot
04/2006	Nugache	P2P bot
01/2007	Peacomm	P2P bot based on Kademlia
10/2007	Storm	Uses its own P2P network

Table 2.1: The Timeline of Bots

2.2 The Threat of the Botnets

The primary goals of botnets can be categorized as information dispersion and information harvesting. Information dispersion includes the e-mail spamming attacks and denial of service attacks. Information harvesting aims to obtain identity data, financial data, private data, e-mail address books or any type of data may exist on the host. Although some of the botmasters construct their botnets for fun or fame, most of them intent to get financial benefits. The information dispersion has economic benefits because some companies may wish to pay the botmaster in order to disperse spams that are used for sending advertisements. The information harvesting also has direct economic benefits, since the revealed secret information may allow the botmaster to get money directly.

Today, botnets constitute a big treat against Internet users. They perform malicious activities that aim to steal important secret information, obstruct working of a system, make advertisements or send junk e-mails, etc. The most popular and effective attacks performed by botnets are distributed denial of service attacks, e-mail spamming attacks and phishing attacks. In addition, the botnets might also be used for identity theft and click fraud as well.

2.2.1 Distributed Denial of Service Attacks

Botnets are widely used to perform distributed denial of service attacks (DDoS), which can be significantly destructive if the size of the botnet is big enough. A DDoS attack is an attack that targets either a computer or a network to make a resource unavailable to its users. Typically, the loss of the service or network connectivity is done by consuming the bandwidth of the network or overloading the network stack in the computer. A DDoS attack can be performed in a number of different ways. Some of them are listed as:

- Consuming the computational resources, e.g. bandwidth, disk space or processor time.
- Corrupting the configuration information such as routing table configuration.
- Disrupting the state information, such as unsolicited resetting of TCP sessions.
- Corrupting the physical network components
- Obstructing the communication media in order to prevent the users from communicating each other.

Today, it is very easy to mount DDoS attacks with the help of off-the-shelf tools [8] . There are different kinds of attacks that target the connection oriented Internet protocol TCP, the connectionless protocol UDP or protocols at higher level in the network stack:

1. *TCP SYN flooding :* TCP SYN flooding attack is performed by sending several connection requests to target computer in order to stress the processing ability. The half open connections on the target machine exhaust the data structures in the kernel. Thus, the computer cannot accept new connections.

2. *UDP flooding :* The attacker aims to consume the network bandwidth and computational resources by sending a large number of UDP packets to several ports. While the UDP flooding attack can be used to perform DDoS attacks, the attacker can get some important information, such as the services working on specific ports, as well.

3. *DDoS attacks targeting high-level protocols:* Anymore, the DDoS attacks are more dangerous, because they are not only restricted to web services. By creating more specific attacks that target high-level protocols, more efficient results can be obtained. The *web spidering attack*, which starts from a given web site and then recursively requests all links on that site, is a good example for DDoS attacks that targets high-level protocols.

In the past, several serious DDoS attacks were seen. In February 2000, an attacker applied DDoS attacks to several e-commerce companies and web sites. The attacks deactivated the service of the servers for several hours. In recent years, the threat of the DDoS attacks turn into real cybercrime. For example, a botnet targeted a betting company during the European soccer championship in 2004 and demanded money in exchange of letting the system operate again.

2.2.2 E-Mail Spamming

E-mail spamming, a.k.a. bulk e-mail or junk e-mail, is to send nearly identical messages to numerous recipients by e-mail. Generally, such messages have commercial content. An e-mail is spam only if it is unsolicited and sent in bulk. E-mail spams slowly but exponentially can grow to several billion messages a day. Thus, now e-mail spamming is one of the most disturbing Internet activities. E-mail addresses used by the spammers are collected by chat rooms, newsgroups, websites and the malware that harvest e-mail addresses from the users' address books.

The 80% of the spam e-mails are sent by the botnets. Typically, bots start a SOCKS v4/v5 proxy on the compromised host in order to use it for sending spam e-mails. Obviously, a botnet with thousands of members can send a massive amount of spam e-mails.

While mostly the bots are known to send spam mails, there are other kinds of bots that are specifically used for collecting valid e-mail addresses from the wild. Most of the spambots are used for collecting user addresses. Such spambots are web crawlers that can gather e-mail addresses from Web sites, newsgroups, special-interest group (SIG) postings, and chat-room conversations.

2.2.3 Phishing Mails

Phishing is a kind of identity theft which aims to compromise sensitive information, such as passwords or credit card information, by masquerading as a trustworthy entity in an electronic communication. Now, phishing attacks use sophisticated social engineering techniques to persuade users to give their secret information. There are different types of phishing attacks:

- *Spoofing Mails and Web Sites :* The earliest phishing attacks were e-mail based. The attackers were trying to persuade the victim users to send their passwords and account information by sending spoofed e-mails. Although there are still many users that can be fooled, anymore most of them know that sensitive information must not be sent by e-mails. Thus, the attackers developed more sophisticated phishing techniques to deceive the victims. One of the well-known phishing attacks combines both phishing mails and web sites by sending mails that appear to come from a legitimate organization. After the user clicks a link in the mail, the e-mail directs the user to a web site that looks identical to a familiar web site. Then, the user perform his normal actions, such as logging into the site or sending account information, which reveals all the secret information to the attacker.
- *Exploit Based Phishing Attacks:* Exploit Based Phishing Attacks are more sophisticated. They make use of known vulnerabilities to exploit the system and then install a program that collects the sensitive information. A good

example for such programs is key-loggers which records all of the keys that are pressed by the user in order to get secret information.

2.3 Characteristics of Botnets

The attributes that characterize bots are the remote control facility, the command set that is used for several nefarious purposes and the spreading mechanism to increase the population of the botnet. The remote control facility allows the attacker to have full control over the infected machines. The remote control mechanisms can be either centralized or decentralized. Centralized control mechanisms are divided into two categories: push style and poll style. There is only one example for decentralized control mechanisms, which is used by the peer-to-peer botnets. The command set defined for a botnet may comprise a wide range of commands that intends to compromise important data, such as secret information or e-mail address books, attack a target machine, send spam mails etc. Generally most the botnets focus on implementing commands that lead the bots to perform DDoS attacks or update themselves.

While remote control mechanism and commands differentiate bots from worms, they have similar spreading mechanisms as worms have. Usually, in order to propagate, the bots automatically scan some specific network ranges. If they can find any vulnerability, they exploit it and afterwards, copy themselves to the victim machine. Since machines that have Windows operating system have so many vulnerabilities, bots generally attack Internet users who use Windows operating system.

The attributes that distinguish the bots and characterize different bot families are bot propagation mechanisms, command and control mechanisms, exploit and attack mechanisms and obfuscation mechanisms. In the following sections, we give detailed information about characteristics of bots.

2.3.1 Bot Propagation Mechanisms

The more compromised machines, the more effective the botnet is. Thus, propagation of bots is a necessary step in bots' lifecycle. Propagation refers to the mechanism used for finding new vulnerable machines to take their possessions. To this end, bots simply make use of some traditional scanning mechanism, such as horizontal scanning or vertical scanning. Horizontal scanning mechanism scans a single port in a specified address space, and on the other hand vertical scanning mechanism scans a port range on single IP address. Since the main purpose of the propagation is to infect machines as many as possible, to date, more sophisticated propagation mechanisms have developed. Obviously, the botnet designers adopt the strongest and the most efficient scanning schemes to their systems to expand their capabilities.

The most well-known scanning mechanisms are:

- *Random Scanning:* The target to be scanned is determined by a random number generator. Thus, efficiency of the scanning strictly depends on the random number generator. Since it is quite difficult to develop a random number generator that can find vulnerable hosts or valid IP addresses, the random scanning is not effective enough.
- *Permutation Scanning:* The random scanning is inefficient because it produces overlaps. Permutation Scanning was designed to deal the problem of overlaps at the random scanning. It makes use of simple cryptography to make different malware samples generate different addresses. Simply, all of the malware samples share a common pseudo random permutation and use a private key to generate the addresses. Therefore, permutation scanning relatively solves the overlapping address problem.
- *Hit-List Scanning:* It is a very fast method. However, since the whole hit-list comes within the malware binary, the binary is very big. While the binary is spreading, the size of it gets smaller. Because, the binary scans only first n addresses in the list, and when it finds a vulnerable host, only

sends the remaining part of the list not all. Clearly, the reason for not sending whole of the list is to avoid overlapping.

- *Combining the Techniques:* Some of the worms seen in the wild, such as *Warhol* worm, uses the combination of permutation scanning and hit-list scanning methods. The method is capable of attacking whole of the vulnerable machines in less than fifteen minutes.

Although there are botnets that use very sophisticated scanning methods, very well-known bots, such as Agobot, SDBot, SpyBot and GTBot, still have simple propagation schemes that consist of vertical and horizontal scanning. This means that it may be possible to develop statistical finger printing methods to identify bot scans. The only advantage of the bots over the worms is that the botmaster can specify and change the address ranges that are randomly scanned when she notices that the address range scanned is invalid or does not have any vulnerable host.

2.3.2 Command and Control Mechanisms

Command and Control (C&C) mechanism refers to the command language and control protocols used for managing the botnets remotely. The C&C mechanism is the strongest attribute of the botnets, since the botmaster can define a command set for her intentions. Moreover, if there is also an updating mechanism, she can modify the command set by adding new commands or removing the ones that are not necessary anymore. That is to say, the C&C brings a great flexibility to the activities that can be performed by the bots. Nevertheless, C&C is also the weakest link of the system. Thus, in order to find detection models for botnets, the C&C mechanisms have to be analyzed in detail.

The common command and control infrastructure that is used for managing the botnets is based on Internet Chat Relay (IRC): The attacker sets up a private channel on an IRC server for her own purposes. The bots connect to that channel, and behave according to the commands that are sent. Some of the attackers use

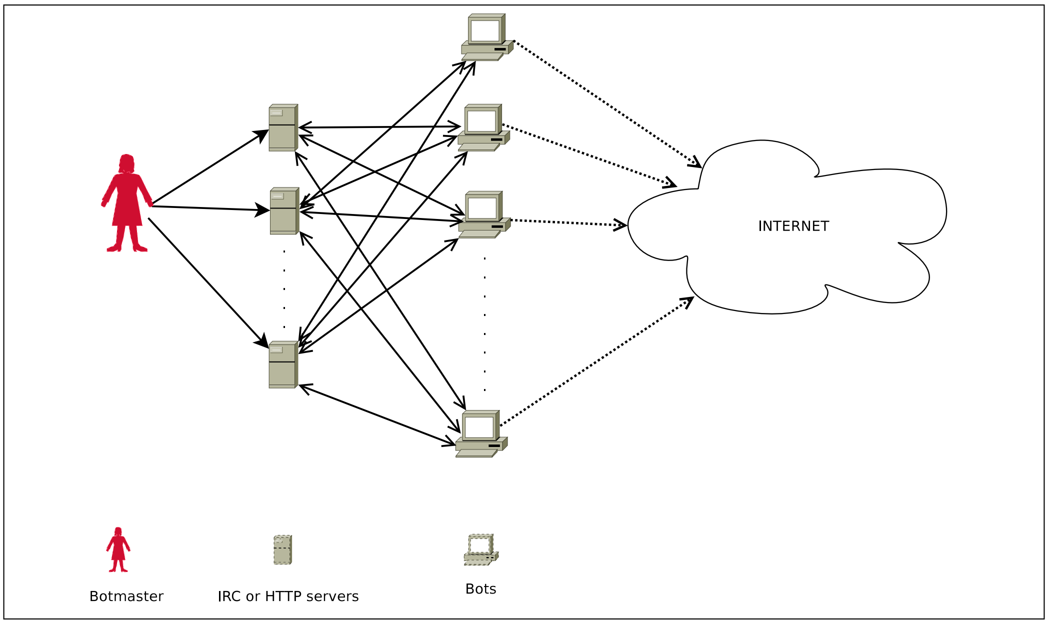


Figure 2.1: Botnets that have centralized command and control mechanisms

an HTTP server for commanding their bots. Obviously, the members of this setup are called HTTP bots, since the C&C protocol is HTTP. Contrary to IRC bots, HTTP bots do not connect to a channel and wait for the commands. They periodically poll the server for new commands and act upon them. Although HTTP bots and IRC bot differentiate from each other at the way of getting the commands, both of them can be categorized as centralized botnets, since they both get the commands from a central point. The Figure 2.1 shows the general structure of centralized botnets. Lately, a new generation of botnets that use P2P style communication appeared in the wild. Such botnets do not have any centralized server that distributes the commands. Instead, all bots in the botnet behave both like a server and a client. Thus, the C&C mechanism that are used by P2P botnets are decentralized.

2.3.2.1 Push Style C&C

A typical setup for a botnet that has a push style C&C is shown in the Figure 2.2. A central IRC server is used for the C&C, some of the botnets have more than one IRC server as shown in the Figure 2.1. The reason for using multiple servers

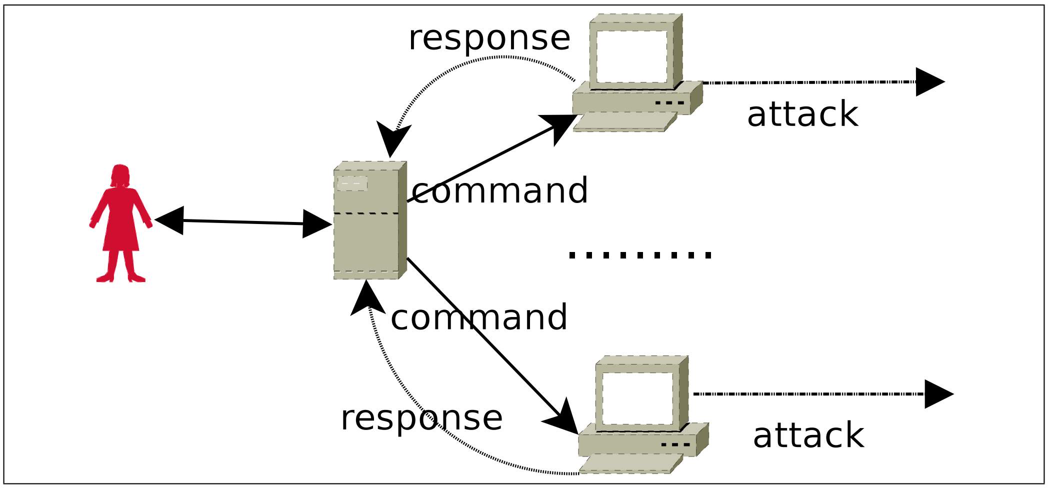


Figure 2.2: Push Style C&C Mechanisms

to spread the commands is to continue the malicious activity even one of the C&C servers is shut down or noticed by botnet trackers.

As soon as the bot binary is run in the victim machine, the bot connects to an IRC server at a specific port and afterwards joins a predefined channel. The attacker releases the commands from that channel and the bot acts as if it is commanded. For example, if the botmaster sends a command that demands the bots to do denial of service of a target machine, the bots start sending several packets to the target. The time that the bots will stop the attack may be specified either in the attack command or with in a new command that demands the attack to be stopped.

Commands can be sent to the bots in several different ways:

- When the botmaster wants to send a command to only one bot, she can send the command via *PRIVMSG* IRC command that has the bot's user name as a parameter.
- The *PRIVMSG* command can also be used for sending a broadcast command that will lead all of the connected bots act simultaneously. This is done by passing the channel name as a parameter instead of a specific bot name.

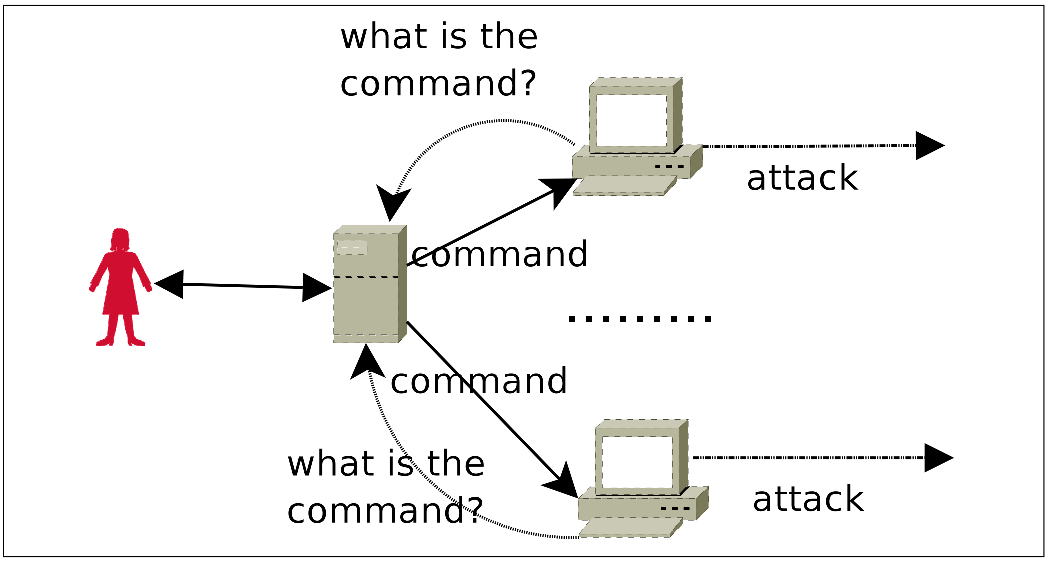


Figure 2.3: Poll Style C&C Mechanisms

- The channel's topic name can be used to send the command to all of the bots as well. When the botmaster wants to send the command, she simply changes the topic of the channel by the *TOPIC* IRC command.

If the topic of the channel does not have any instruction, the bots are idle in the channel, waiting for the commands. Another important issue that has to be mentioned is authentication that should be done by the botmaster to have a full control over the bots and the channel. Since the botmaster creates the channel, she is the owner and has the rights to do whatever she wants. The IRC servers require a username and password to authenticate the members. Thus, it is enough to enter the correct username and password to get the full control to start the malicious activities.

2.3.2.2 Poll Style C&C

In contrast to push based IRC C&C, HTTP bots use a poll based system. The HTTP bots' C&C mechanism is called poll based because of the periodic queries done by the bots. The botmaster who intends to command her bots by HTTP C&C simply runs a HTTP server that has a specific IP address and places the command to a file that is queried periodically by the bots as it is shown in Figure 2.3.

The poll based C&C mechanism is weaker than the push based systems, since the botmaster does not have a real time control over the bots. That is to say, the command can not be sent unless the bots query the server. Nevertheless, there are well-known botnets, such as Bobax [32], that use the HTTP protocol for command and control.

Since botnets with poll style command and control make use of the HTTP protocol, the bots query the HTTP server with the *GET* command. Generally, they also send their status information within the request. The status information may consist of the ID of the bot, the operating system running on the victim machine, information about the connection type, the local time of the compromised machine, etc.

2.3.2.3 P2P C&C

Botnets that have a peer-to-peer(P2P) structure are not managed in a centralized manner. Thus, the C&C mechanism of such botnets is called decentralized. The nodes in the botnet behave as both a server and a client. Therefore, the botmaster can not be easily caught. Compared to botnets that have centralized C&C, it is more robust. Because, even if some of the nodes in the network are shut down, the gaps in the network are closed and the network continues its activities under the control of the botmaster.

Most of the well-known P2P botnets use Overnet network. The Overnet is a Kademlia based protocol, which provides a method to locate values that correspond to given search keys. The bots do not directly send information to each other, instead when the botmaster wants to send a command, she publishes a piece of information i, using an identifier derived from the information. Every day to get the commands, the bots search for 32 different keys, which are computed with a function that takes the current date and a random number between 0 and 31 as a parameter. Since the attacker knows which keys are searched every day, she publishes the command under one of those keys. Basically, after the command is received by the bot, it starts the malicious activities.

2.3.3 Exploit and Attack Mechanisms

Since the botmaster wants to propagate whole over the Internet in order to increase the population of her botnet, she applies some propagation strategies as explained in Section 2.3.1. There are some different ways to compromise the victim machine, such as exploiting a known vulnerability or deceiving the user of the computer to click a link that might be sent via a chat program, a mail or a phishing site. Infecting a machine by deceiving the user is quite simple, since immediately after the user clicks the link, bot binary is downloaded and run. On the other hand, infecting by exploits needs elaborate work. To compromise the machine, firstly the exploit code that uses the vulnerability must be developed. Then, in order to find machines that have the vulnerability, a scanning mechanism has to be specified. Finally, to get the full control over the machine, the exploit has to be applied.

Typically, the sophisticated bots, such as Agobot [referans], develop exploits for several vulnerabilities. Clearly, if the bot has more than one exploit, it can infect more vulnerable machines. Agobot has the exploits listed below:

1. *Bagle scanner:* Scans for backdoors on port 2745.
2. *Dcom scanner:* Scans for DCE-RPC buffer overflow.
3. *MyDoom scanner:* Scans for backdoor on port 3127.
4. *Dameware scanner:* Scans for the Dameware network administration tool which is vulnerable.
5. *NetBIOS scanner:* Brute force password scanning for open NetBIOS shares.
6. *Radmin scanner:* Scans for Radmin buffer overflow.
7. *MS-SQL scanner:* Brute force password scanning for open SQL servers.

The most destructive attack performed by the botnets is the distributed denial of service attack. Thus, most of the bots have the DDoS attack implemented. Agobot is able to perform seven different types of DDoS attacks: UDP flood, SYN flood, HTTP flood, PHAT SYN flood, PHAT ICMP flood, PHAT WONK flood, targa3 flood.

2.3.4 Obfuscation Mechanisms

Obfuscation is defined as “The concealment of meaning in communication, making it confusing and harder to interpret.” in Wikipedia. Thus, the term $obfuscation$ refers to the mechanism to hide the commands that are sent by the botmaster.

Formerly, almost all of the botnets used clear text protocols that did not hide the communication traffic between botmaster and bots. After the threat of the botnets was realized, the researchers started to look for ways to detect and prevent botnets’ malicious activities. They did reverse engineering of the C&C protocols to produce signatures that can be deployed on the vantage points of the networks. Obviously, in order to make their system undetectable, attackers obfuscated the commands with some predefined keys. Anymore, it was difficult to understand the content of the command and for what reason it was sent. Fortunately, they did not estimate that the obfuscation they applied was useless unless they change the key each time the command is sent. When they always use the same key to obfuscate the command, even though it is difficult to transform it to the clear text, the obfuscated command could still be used to generate a signature. Of course, the attackers who have got a sophisticated knowledge about encryption use strong encryption techniques to obfuscate the C&C. To date, none of the detection models are able to detect botnets that uses encryption.

2.4 Real World Examples for Botnets

2.4.1 IRC Bots

The most prominent IRC bots are *Agobot, SDBot, SpyBot* and *GtBot*. We will take a closer look at *Agobot*, which is the most sophisticated IRC Bot that has several advanced features, and *SDBot*.

- *Agobot:* Agobot is the best-known family in all of the IRC Bots. It has several variants, such as *Phatbot*, *Forbot* and *XtrmBot* and the antivirus vendors claim that there are 1500 more. Agobot was published in 2004 and pretty soon after it, so many variants started to appear in the wild. The code of Agobot, which has a very high abstract design that allows adding new features such as new commands or new scanners for new vulnerabilities, was written in C++. Although most of the Agobot variants use an IRC server to set up the C&C mechanism, some of them use peer-to-peer to protocols to construct a decentralized C&C mechanism. Typically, Agobot variants have more than one spreading, DDoS attack or update mechanisms and since they have abstract design, always it is possible to add more. Moreover to the features about the bot characteristics, they have features to kill the antivirus programs or malware monitoring systems installed on the infected machine. Agobot and its variants use the packet sniffing library *libpcap* to sniff the traffic passing through the network adapter of the victim machine. They use NTFS Alternate data stream to hide the malware and offer rootkit capabilities. The reverse engineering of the binary is almost impossible, since they use functions to detect debuggers and encrypt the configuration files of the binary. Agobot binary activates itself just after doing a speed test for Internet connectivity. They connect to a specific server, then send and receive data. This feature of Agobot reveals information about the count of the machines infected by Agobot. In 2004, 300.000 unique IP addresses were identified per day.

- *SDBot:* Most of the active bots that are seen in the wild are either SDBot or its variants, such as *RBot, UrBot, UrXBot* and *SpyBot*. Since its source is public too, it has several variants as well. The source code does not have a good design as Agobot. Nevertheless, most of the botmasters use its code. It provides a rich set of features as Agobot provide. Most of the SDBot variants use the IRC C&C, however there are some that use HTTP C&C mechanisms. Currently, identity theft and stealing sensitive information is a big threat against Internet users. Spybot, which is a variant of SDBot, provides a rich command set to get sensitive information about the compromised machines.

2.4.2 Storm

The most famous P2P bot currently spreading in the wild is known as *Peacom, Nuwar* or *Zhelatin*. Because of its devastating success, it was given the name *Storm worm*. Unlike the worms and all common IRC bots, which propagate by exploiting remote code execution vulnerabilities in the network services, storm worm merely propagate by using e-mails. The e-mail body contains a text that tries to deceive the user to click a link or open an attachment. If the user is deceived, the malware is downloaded to the users machine. The propagation vector of storm botnet is analyzed with the help of *spamtraps*. The reports on the spamtrap archives show that storm is quite active and can generate a significant amount of spam, which is 10% of the spam generated whole over the world.

Storm worm has a sophisticated malware binary, since it uses several advanced techniques. Each time the storm binary is downloaded from the same source, the size of the binary changes and it means that storm worm uses a kind of polymorphism. Moreover, the binary packer that is used by the storm is the most advanced seen in the wild and it uses a rootkit in order to hide its presence on the infected machine.

The first version of Strom uses OVERNET, a Kademlia-based [21] P2P distributed hash tables(DHT) routing protocol, as the C&C mechanism. In October

2007, Storm botnet changed its communication network from OVERNET to its own P2P network, which is called as *Stromnet*. The new network is identical to OVERNET except for the fact that each message is XOR encrypted.

In order to find other infected peer, the storm bot searches for specific keys that help the bot to distinguish between regular and infected peers in the OVERNET. The key is generated by a function $f(d, r)$, where d is the current day and r is a random number that takes values between 1 and 32, thus there can be 32 different keys per day. Since the botmaster is aware of the keys that are searched by the bots every day, she issues the commands to specific keys. If the command is issued to a key, the search result of the key is the command that triggers a kind of attack behavior, such as e-mail spamming.

In order to track Storm botnet, the researchers leverage the fact that some specific keys are searched every day [12]. As a result of the research on storm botnet tracking, they estimate lower and upper bound for the count of storm infected machines. Their assertion is that the lower bound is about $5.000 - 6.000$ and upper bound is about $45.000 - 80.000$.

Chapter 3

System Overview

The aim of our system is to develop network-based detection models in order to identify bot-infected machines. We take the network traffic captures that are produced by real bots as input, and as the output we create couple of detection models that are capable of detecting infected machines in a network.

Our bot detection model consists of three states. The first state is called the idle state where bots perform nothing but wait idle for the commands. The detection model switches to the second state only if a command that is sent by the botmaster is matched. In other words, the second state indicates that the command, which may result the bot to start a denial of service attack, spamming or other malicious activities, is received by the bot. We expect that immediately after the command is received, the bot performs an activity as a response. Thus, the detection model switches to the third state if a response behavior is detected. Typically, the bots react immediately thereafter the command is received, however there are exceptions. Therefore, if the response activity is not observed in t seconds, the detection model switches to the idle state again. The experiments showed that the time threshold of 100 seconds is reasonable to observe the response.

We give a system overview in the following sections that explain the phases of the system in an ordered fashion.

3.1 Running The Bot Samples

Since the input of our system is network traffic captures that are likely to have the network activities performed by real bots, we had to run the bot samples in a controlled environment that allows the bot binaries to establish connections with the botmaster. To this end, we run each bot binary in the controlled environment for a period of several days and capture the network traffic produced in that period. The bot binaries are collected in the wild, for example, via honeynet systems such as Nepenthes [1], or through Anubis [3], a malware collection and analysis platform. Thus, the network traces analyzed by our system have the real command and control traffic produced by real bots and real botmasters.

The bot binaries are run in VMware [34] virtual machines that have fully-patched Windows XP with service pack 2 for a period of time. At the same time, in order to prepare the input of our system, the network traces that are produced by the bots are captured and recorded. Approximately, the running duration of the virtual machines averages out of four days, which is an empirically chosen time duration that makes it possible to observe the bot commands and responses. The Chapter 4 gives more detail about the phase where the network traces are captured.

3.2 Clustering Bot Families

We define a bot family as a set of bots that use the same command and control mechanism and perform similar responses to the similar commands. That is to say, a command of a specific family always triggers the same behavior on the bots that are members of the family. Therefore, different bot families use different commands in order to perform their malicious activities. For example, while an IRC bot uses *.advscan* to start a scanning activity, the other one uses *asc*.

A signature that is responsible of detecting a specific behavior can be generated only if the commands that trigger the behavior have some commonalities.

Since different bot families use different commands, the bot samples that produce the input of our system have to be partitioned to construct different bot families. This partitioning can be performed either manually, based on malware names assigned by virus scanners, or automatically, based on behavioral similarities that are observed when the malware is run in host-based malware analysis systems. We have made use of a malware clustering system, which is an extension to Anubis that analyzes the execution traces of malware to find behavioral similarities. However, the clustering of the bot families needs some manual effort too. We are not responsible for making the perfect clustering of the bots which is rather a prerequisite step for our system.

3.3 Finding Bot Responses in the Network Captures

Since the bot responses are more visible than bot commands that are generally sent in some short TCP or UDP packets, instead of tracing the signs of bot commands in the network traffic captures, we try to locate the bot responses. In order to identify the behavioral changes in the network traffic, we analyze the captures relying on some network properties, as described in Chapter 5. The network properties that are able to identify the behavioral changes are; the count of packets, the total size of the packets, the count of packets that are sent by couple of specific protocols, the count of non-ascii characters and count of packets that have unique ports or ip addresses. As long as the botmaster does not perform *time-bombs*, a bot command is followed by a bot response in a certain amount of time. The experiments show that generally the bot response is performed in maximum 100 seconds after the command is issued. Thus, we cut the 100 seconds long traffic capture that precedes the bot response. Since these network snippets are likely to include the bot commands, we use them as the input to the signature generation algorithm which tries to find common tokens within the snippets that belong to different bot samples.

3.4 Extracting Behavioral Profiles

We assume that network snippets that are extracted from bot samples, which belong to the same bot family, and lead to the same response, contain relevant commands. Clearly, different reactions should be caused by different commands. For example, the network snippets that are followed by a scanning behavior should include the commands that lead the bots to perform scanning activity, not the others. Obviously, if we can group the network snippets that have the same behavioral profile, we may extract some commonalities that allow us generate signatures that are able to detect the bots that performs a specific behavior.

In order to gather related network snippets together, we use a clustering algorithm [7]. The hierarchical clustering algorithm clusters the network snippets according to their behavioral profiles. The clustering is stopped when the minimal distance between any two clusters exceeds a threshold. Once the clustering step is finished, the snippets in each behavioral cluster are ready to be analyzed for finding common tokens in order to construct the signatures.

3.5 Generating Signatures

The last step of our system is signature generation, as described in Chapter 6 in detail. We generate token subsequence signatures from the command tokens observed in snippets that have the same behavioral profile. Since our signatures can also be defined as regular expressions, they can easily be deployed to the well-known intrusion detection systems, such as Bro and Snort. As we have mentioned before, our detection scheme has three states. The signature generation phase outputs the signatures that can be used for the second state. For the third phase, we leverage the knowledge that we get from the behavioral profiles. In other words, in our current system, a bot detection model consists of a set of tokens that represent the bot command, followed by a network-level description of the expected response.

Chapter 4

Experimental Setup

In order to produce signatures, which are used for detecting bot-infected machines, we need to analyze network traffic produced by bots. Thus, the first step to be accomplished in our project is to capture network traces that have the communication between bots and the botmaster. To this end, we run each bot binary in a controlled environment for a period of time and capture the network traffic produced in that period. The bot binaries mentioned above are collected in the wild, for example, via honeynet systems such as Nepenthes [1], or through Anubis [3], a malware collection and analysis platform.

In the following sections, we explain how the bot binaries are collected, how the environment that is used for running the bot binaries is built and how we capture the network traces in detail.

4.1 Collecting Bot Binaries

Malware is software designed to infiltrate or damage a computer system without the owner's informed consent. There are different kinds of malware seen in the wild such as computer viruses, worms, Trojan horses, rootkits, spyware, dishonest adware etc. The most dangerous ones are the malware which can spread all over the network by jumping from one machine to another. Unfortunately, the treat of malware is not against only the individual computers, but more important networks. Especially thereafter botnets occurred in the Internet; they started to control thousands of computers for their nefarious purposes such as performing denial of service attacks to crash a target system.

To construct a defense against malware, intrusion detection systems and antivirus systems analyze the malware samples. Then, they make use of the analysis results to generate signatures that identify particular malware. Of course, collecting the malware and analyzing it is not a trivial task, especially if they require manual effort. Thus, to provide high degree of automation on these steps, honeynet systems are developed. Today, the most popular malware collecting technology is Honeypot technology. A honeypot is a trap set to detect, deflect, or in some manner counteract attempts at unauthorized use of resource. Honeypots can be classified on their level of involvement:

- *Low-interaction honeypots* simulates only services that cannot be exploited to get complete access to the honeypot, which makes the risk of being compromised very low. Generally, they simulate one part of the operating system such as the network stack. The *honeyd* [26] is a good example for this kind of honeypots. Although low-interaction honeypots are more limited, they are useful to gather information at a higher level, e.g., learn about attack patterns, propagating vector.
- *High-interaction honeypots* simulates all of the aspects of an operating system. Thus, the attacker can compromise whole of the system and launch her attacks without any restriction. High-interaction honeypots allow the analyzer to study the attacker's behavior in more detail. The most common example for this kind of honeypot is *Honeynet* [2].

The bot binaries analyzed in our project is collected via Nepenthes and Anubis, which are explained in more detail below.

4.1.1 The Nepenthes Platform

Nepenthes is a low-interaction honeypot which has a high degree of expressiveness. It is not a honeypot by itself but a platform to deploy honeypot modules that are called vulnerability modules. The nepenthes can be easily configured by vulnerability modules into a honeypot for many different types of vulnerabilities. The flexibility of nepenthes allows deploying features that are not impossible to be analyzed by high-interaction honeypots. For example, since emulation can mimic general traffic patterns of network communication to behave either like Linux or Windows, nepenthes can emulate the vulnerabilities of different operating systems or architectures in a single machine or during a single attack.

Nepenthes platform is able to collect malware that is currently spreading in the wild on a large-scale. The HTTP bot and some of IRC bot binaries that are used in our experiments are mostly collected by nepenthes.

4.1.2 ANUBIS:Analyzing Unknown Binaries

Anubis is a public service for analyzing Windows executable binaries. The binaries that are analyzed daily by Anubis are either collected by honeypots or spamtraps, or submitted by public users. It tries to extract behaviors of the executables with special focus on analysis of the malware. To this end, the binary executable is run in an emulated environment and its security-relevant actions are monitored.

The features analyzed by Anubis are;

- *Analysis of Registry Activities*
- *Analysis of File Activities*
- *Analysis of Process Activities*
- *Analysis of Windows Service Activities*

- *Analysis of Network Activities*
- *Native API aware Analysis*
- *Unobtrusive Analysis*
- *Complete View of the PC System*

Anubis distinguishes itself from other malware analysis platforms such as Norman Sandbox [23] and CWSandbox [35] with the last two features listed above. Latest malware samples that are seen in the wild check the running environment to find out whether it is a virtual machine or not. Then, they behave differently according to the result of the test to thwart detection. While Norman Sanbox and CWSanbox can be detected by the simple redpill program [29] that checks for the presence of VMWare, Anubis passes the test without detection.

The next generation malware analysis platforms will not be confined to monitor API calls and have complete view of the PC system by analyzing CPU register values and tracking memory accesses. At the present, none of the malware analysis platforms mentioned above has this ability. Anubis is designed to be extensible to the requirements that are possible to appear in the future.

Most of the IRC bot binaries that we experiment and analyze are collected by Anubis.

4.2 Running the Bot Binaries

To create bot detection models, our system requires analyzing the network traffic generated by actual bots. To this end, we run each of the bot binaries in a controlled environment for several days. The goal is to collect enough amount of network traffic that consists of commands that are sent by the botmaster and the responses of the bot. Our experiments show that most of the bot samples have commands that trigger responses within five days. Thus, all of the binaries are run for five days.

Obviously, the more bot binaries are run, the more diverse set of commands are found. Thus, it is necessary to design the execution environment to support running as many parallel bot instances as possible. One approach could be starting several bot binaries inside a single operating system. However, we prefer to start each binary on its own operating system, since there is a possibly to appear an interference between different malware. To this end, we have large number of operating system installation that run parallel in the same machine. The powerful server, which has Intel Xeon 1.86GHz Quadcore processors, 8 GB of memory, and 300 GB of Raid5 disk space, makes it possible to run several virtual machines at the same time.

Unfortunately, latest malware samples check the running environment to find out whether it is a real computer or a virtual machine. According to the result, they behave different. Since, we need to observe actual behaviors of bots; it is a crucial task to choose the most appropriate virtual machine environment. We have analyzed VMware [34], Xen [36] and Qemu [27] that are briefly described in the following sections.

4.2.1 Virtual Machine Monitors (VMMs) and Emulators

Virtual Machine Monitors (VMMs) and emulators provide simulation of hardware so that, the guest software can run in it as if it is executed in real hardware. Popek and Goldberg [25] define a virtual machine as "an efficient, isolated duplicate of the real machine" and specify three the key characteristics of it as:

- *Equivalence*: The software running in the VMM should equivalently perform all of the possible actions that can be done in the real environment.
- *Resource control*: The VMM must be in complete control of the virtualized resources.
- *Efficiency*: Statistically a big amount of the instructions could be executed directly in the hardware without VMM interception. Furthermore, there must not be minor decrease in the running speed.

The third characteristic distinguishes the emulators and VMMs, since emulators do not execute code directly on hardware without interception unlike VMMs. Thus, the emulators cause a decrease in speed.

4.2.1.1 VMware

The VMware consists of a layer of software that is directly on the host operating system. This layer creates virtual machines and contains a VMM that manages hardware resources dynamically and of course transparently so that multiple operating systems can run concurrently on a single physical computer.

VMware introduces full virtualization of x86 systems, in order to transform them into general purpose, shared hardware infrastructure that offers full isolation, mobility and operating system choice for application environments. In this way, VMware virtual machines become highly portable between computers, because every host looks nearly identical to the guest. VMware supports guest operating systems for Microsoft Windows, Linux, Sun Solaris, FreeBSD, and Novell NetWare.

4.2.1.2 Qemu

Qemu is an open source PC emulator written by Fabrice Bellard. To achieve a high execution speed, it relies on a dynamic binary translation. Basically, the dynamic translator converts the target CPU instruction to host instruction set at runtime. The dynamic translation works in terms of basic block. The idea is to translate the code block by block, and execute the block after each translation is done. Obviously, the reason of doing dynamic translation is that it is more efficient to operate one block instead of only one instruction.

Generally, it is difficult to port dynamic translators because the code generator must be re-implemented for each new system. Qemu has a simple and efficient solution that is accomplished by just concatenating pieces of machine code generated offline by the GNU C Compiler [9].

In conjunction with CPU emulation, Qemu also provides a set of device models, allowing it to run a variety of unmodified guest operating systems, thus it can be viewed as a hosted virtual machine monitor. It also provides an accelerated mode for supporting a mixture of binary translation (for kernel code) and native execution (for user code), in the same fashion as VMware Workstation and Microsoft Virtual PC. Qemu can also be used purely for CPU emulation for user level processes; in this mode of operation it is most similar to valgrind.

4.2.1.3 Xen

Xen is a free virtual machine monitor that works in x86 architectures [37]. One of the most important properties of Xen is its *para-virtualization* capability. *Para-virtualization* provides a software interface to the virtual machines which is similar but not identical to the underlying hardware. *Para-virtualization* performs very high performance, even on architectures that are not easily virtualized. This approach requires the kernel of the operating system to be modified and ported to run Xen.

Xen has a multi-layered structure in which the lowest and most privileged layer is reserved for Xen itself. Since the aim of this design is to host multiple operating systems, Xen manages each of the operating systems in different virtual machines, which are called domains. Domain 0 is created automatically for privileged management purposes. Domain 0 creates other domains and manages their virtual machines.

4.2.2 Running Environment: Virtual Machine

The collected bot binaries are run in fully-patched Windows XP with service pack 2. To avoid traffic generated by the operating system, the automatic Windows update is disabled, as well as the Web Proxy Auto-Discovery (WPAD), which causes noisy HTTP traffic during our experiments. By further removing unnecessary Windows components, each bot instance is able to run with 64 MB of main memory each. Using this setup, we are able to run up to 50 virtual machine instances simultaneously on our server.

We created three identical windows XP images that work in Xen, Qemu and VMware. Then, in order to make the system work with the images, we installed additional packages to the host machine that has Debian with 2.6.18-5-686-bigmem kernel. We installed VMware Server 1.0.4, Qemu PC Emulator 0.8.2 and Xen 2.6.18. VMware and Qemu basically install new libraries and tools to start the images. However, it is not so easy to run Xen since we need to modify the kernel.

The next step was to test redpill program in the images that run in VMware, Qemu and Xen. The results are given in Table 4.1. As it can be seen, Redpill could detect only VMware not the others.

VMMs	Redpill test
VMware	detected
Xen	undetected
Qemu	undetected

Table 4.1: The Virtual Machines detected and undetected by the Redpill program

Redpill is a program, as seen in Figure 4.1, that simply checks the address of the interrupt descriptor table. To avoid the confliction, virtual machine monitors move the interrupt descriptor table of the virtual operating system to another safe place in the memory. Thus, if the malware checks the address of the interrupt descriptor table by using SIDT instruction, it can easily detect the presence of virtual machine. Repill is not able to detect emulated systems, since they intercept the instructions and translate them to a corresponding set of instructions for the host operating system. That is to say, the emulators do not change the interrupt descriptor table's address.

Although the VMware was detected by Redpill, we have chosen to use VMware as the virtual machine monitor because of its graphical management interface that allowed us to administrate several virtual machines at the same time. Moreover, experiments showed that most of the current bot samples do not use virtual machine detection tools.

```
int swallow_redpill () {
     unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
        *((unsigned*)&rpill[3]) = (unsigned)m;
        ((void(*)())&rpill)();
        return (m[5]>0xd0) ? 1 : 0;
     }
```

Figure 4.1: The code segment that checks the IDT location in the memory.

4.2.3 Starting Mutliple Virtual Machines

There are two possible ways to start multiple virtual machines at the same time: First is to copy a template instance 50 times and second is to take the snapshots of a base instance. Since our system has enough amounts of memory and disk, we could choose both of them. We have implemented both schemes, however used the first scheme to collect the traffic used to generate signatures.

VMware keeps the virtual machine in a file that has *vmdk* extension. The virtual machines can be configured by the configuration files, which has the *vmx* extension. In the figure 4.2, the template configuration file is shown.

One VMware virtual machine is created with the code in Figure 4.3. As it mentioned before, firstly the VMware template image is copied to a directory that is reserved for one instance. Then, the configuration file named *Sanbox.vmx* is prepared. The parameters to be configured are:

- Memory Size: 64K memory is reserved for each of the instances.
- Display Name: The name of the bot from the view of virtual machine monitor (a.k.a. host).
- Computer Name: The name of the bot from the view of operating system (a.k.a. guest)
- VMnet Device Name: VMware allows binding only 30 virtual machines' network adapters to the same vmnet device, for example /dev/vmnet0. Since we had to create network adapters for 50 virtual machines, we created one more vmnet device */dev/vmnet*1 and bound last 20 of the network

```
config.version = "8"
virtualHW.version = "4"
scsi0.present = "TRUE"
memsize = "<memsize>"
ide0:0.present = "TRUE"
ide0:0.fileName = "Sandbox.vmdk"
ide1:0.present = "TRUE"
ide1:0.fileName = "auto detect"
ide1:0.deviceType = "cdrom-raw"
floppy0.present = "FALSE"
Ethernet0.present = "TRUE"
Ethernet0.connectionType = "custom"
Ethernet0.vnet = "/dev/<vmnet>"
displayName = "<displayname>"
guestOS = "winxppro"
priority.grabbed = "normal"
priority.ungrabbed = "normal"

ide0:0.redo = ""
ethernet0.addressType = "generated"

ide1:0.autodetect = "TRUE"

ide1:0.startConnected = "FALSE"
tools.syncTime = "FALSE"

checkpoint.vmState = ""
checkpoint.vmState.readOnly = "FALSE"

workingDir = "."

machine.id = "<computername>,<IP>,<md5>"
```

Figure 4.2: The configuration of VMware virtual machine.

adapters to it. Thus, in the configuration, the vmnet device that is used has to be specified.

- MD5: The md5 checksum of the bot binary to be executed in the virtual machine.

- IP: IP address to be assigned to virtual machine.

The last step is to start the virtual machine and boot the operating system in it. As soon as the operating system is booted, a script is executed. This script is responsible of making the network settings such as setting the IP address of the network interface and specifying the default gateway. Then, the script downloads the bot binary that has the MD5 checksum that was passed as an argument via the virtual machine configuration file. Afterwards, it executes the malware and leaves the operating system to it.

```
cp -r template /var/vm/bot$1
cd /var/vm/bot$1/
sed "s/<memsize>/64/g" Sandbox.vmx -i
sed "s/<displayname>/bot$1/g" Sandbox.vmx -i
sed "s/<computername>/bot$1/g" Sandbox.vmx -i

if [ $1 -le 30 ]
     then sed "s/<vmnet>/vmnet0/g" Sandbox.vmx -i
     else sed "s/<vmnet>/vmnet2/g" Sandbox.vmx -i
fi
sed "s/<md5>/$2/g" Sandbox.vmx -i
        sed "s/<IP>/$3/g" Sandbox.vmx -i

vmware-cmd -s register /var/vm/bot$1/Sandbox.vmx
vmrun start ./Sandbox.vmx
```

Figure 4.3: Creating and running a virtual machine.

4.3 Capturing Network Traces

Each of the guest virtual machines is assigned a static, public IP address. As our goal is to capture all possible commands and their responses that are sent either by the bots or the botmaster, restricting the network accessibility would have been a wrong decision. Nevertheless, we have to apply some firewall rules, since attacking the network of the university is not a desirable situation. To this end, we block the traffic between the bot and two networks of Technical University of Vienna (TU1 and TU2). Another undesirable situation would have been allowing bots to propagate at the network that is reserved for our bot samples. Because, the packets sent for infecting our bots would have made noise in our captured traffic that is used to find the commands come from the botmaster. Thus, we block also the traffic that flows inside the local network.

We deployed a firewall named GhostWall [30] in each virtual machine. The firewall rules applied can be seen from the Table 4.2.

Description	Rule	Local IP	Local Port	Remote IP	Remote Port
Gateway	Allow all	any	any	*IP-of-gateway*	any
Host	Allow all	any	any	*IP-of-host*	any
Anubis	Allow all	any	any	*IP-of-Anubis*	any
Broadcast(BC)	Allow all	*BC-IP*	any	*Broadcast-IP*	any
Local Network	Block all	any	any	*Local Network*	any
DNS 1	Allow all	any	any	*IP-of-dns1*	any
DNS 2	Allow all	any	any	*IP-of-dns2*	any
TU1	Block all	any	any	*TU-network1*	any
TU2	Block all	any	any	*TU-network2*	any

Table 4.2: The firewall rules of each VMware virtual machine

The network traces were captured in the host machine during five days. Unfortunately, two times the attackers noticed that we were doing research about botnets. To avoid us getting information about their activities, they tried to crash our host machine by doing denial of service attacks. As soon as we noticed the denial of service behavior in our traffic, we stopped the virtual machines. Thus, we could not run all of the bot binaries for five days. Approximately, the running duration of the virtual machines averages out of four days.

In order to capture the traffic, we make use of tcpdump [33], which can print out the packet contents that pass through a specified network interface. When the appropriate parameters are used, the captured traffic by tcpdump can be written to a file for later analysis. Moreover, the undesired traffic such as ARP packets, the packets generated because of the communication between the host and virtual machine, can be filtered out.

Immediately thereafter the fifty virtual machines are booted, the tcpdump on the host machine starts capturing the traffic into a *PCAP* file. Capturing process continues until the virtual machines are killed. As the tcpdump listens to the traffic passing through the network interface that is bound to all of the network adapters of fifty virtual machines, the *PCAP* file contains the traffic belongs to all of them. Since our aim was to find the commands sent by the botmaster and the reponse of the bot, analyzing the traffic capture that belongs to an individual bot would be reasonable. To this end, the *PCAP* file was later split up according to the IP addresses of the virtual machines. Since there are no other applications that run and generate network traffic, the bot accounts for all observed network traffic under the virtual machine's IP address.

Chapter 5

Analysis of the Network Traffic

The most powerful and also the weakest property of botnets is the necessity for communication between bots and botmaster. This property is powerful, because it improves the malware's ability by adding the property of manageability. Thus, the malware can be controlled by the attacker in order to perform her nefarious purposes. On the other hand, the network traffic generated by the communication makes the system weak against intrusion detection systems deployed at vantage points of network. The reason is that they can easily detect bot-infected machines if they manage to find some commonalities at command and control protocol. The most well-known intrusion detection systems to date are Snort [28] and Bro [24]. Their defense strategy is to monitor the network traffic and alert the network administrator when a predefined suspicious activity is detected. Since our goal is to detect bot-infected machines by analyzing network traffic for presence of signs that indicate the infection, we can make use of such intrusion detection systems. Obviously, wiser botnet designers are aware of the weakest link of their system. Therefore, they find some solutions to elude intrusion detection systems. One possible solution is either to obfuscate or to encrypt the command and control protocol in order to avoid malware analyzers to find commonalities at the communication protocol. Although encryption makes malware undetectable against intrusion detection systems that have a content-based detection scheme, there is an open door to find network-based detection strategies since network traffic of a bot includes command and response behavior all the time.

To gather more information about bots, we have done elaborate analysis of the network captures, since the goal of our system is to create network level detection strategies for bot-infected machines. The analysis consists of counting packets and finding the total amount of data carried by the packets on a time interval that can be tuned to make either more general or more specific analysis. Although it seems too simple, astonishingly it revealed so much significant information about behaviors of bots. Especially, the graphics that are sketched from the data produced by the analysis helped us to observe a diverse set of bot activities.

We have analyzed the network traffic captures with two different approaches. First, to get a general opinion about the communication performed by bots, we sketched graphics for whole capture without focusing on individual connections. As a result of this analysis, we were able to identify activities such as scanning or denial of service attacks. However, without focusing on individual connections it is not possible to get information about the command and control mechanisms. To this end, we have also done connection based analysis. The connection base analysis let us bot activities such as downloads, updates, spamming, etc.

In this chapter, we will not only give detailed description of our network analysis but also present the results of analysis that reveals very interesting activities performed by bots.

5.1 Aggregate Analysis

The idea behind analyzing the traffic captures in a generalized fashion is to find a protocol independent detection scheme. To this end, we analyze all of the flows in the network traffic captures aggregately. Therefore, statistics that is evaluated for a time period includes all of the traffic flows in that interval. Basically, we analyze the network traffic produced by a bot in order to locate commands and their corresponding bot responses in the traffic. Thus, we try to recognize behavioral changes by examining the traffic according to some network properties:

- *Count of packets:* We count number of UDP and TCP packets sent and received. We examine the traffic by counting the packets to detect scanning activities or denial of service attacks.

- *Count of packets that belong to a specific protocol :* We count number of packets that belong to a specific protocol, such as $HTTP$, $SMTP$, FTP etc. , to observe activities carried out in that protocol.

- *Count of different IP addresses:* We count the number of different connections or connection attempts to identify address scanning behaviors.

- *Count of different Port numbers:* We count the number of different destination ports in the packets in order to identify port scanning behaviors.

- *Size of data transferred :* We sum the size of the packets that are carried to identify downloads.

- *The count of non-ascii characters:* We sum number of non-ascii characters in the packets to observe binary downloads.

We have produced statistic data for each property listed above and then, sketched graphics to visualize behavioral changes. The graphics were sketched with three different time intervals, 10, 100 and 1000 seconds. The graphics with 10 seconds time interval was too specific, because very small changes, which prevent us to recognize the significant behavioral changes, are also handled. On the other hand, the graphics that have 1000 seconds time interval were too general that hid some important activities. Thus, we have chosen 100 seconds time interval in our analysis, since it produced the most reasonable values that allowed us see most of the important activities carried out by bots.

The Figure 5.1, Figure 5.2, Figure 5.3, Figure 5.4, Figure 5.5 and Figure 5.6 show the graphics that are sketched from a captured network traffic that consists of an IRC bot's network activities.

The graphic in the Figure 5.1 shows alteration of the packet count in every 100 seconds. Generally, as soon as the bot is run in the infected machine, it

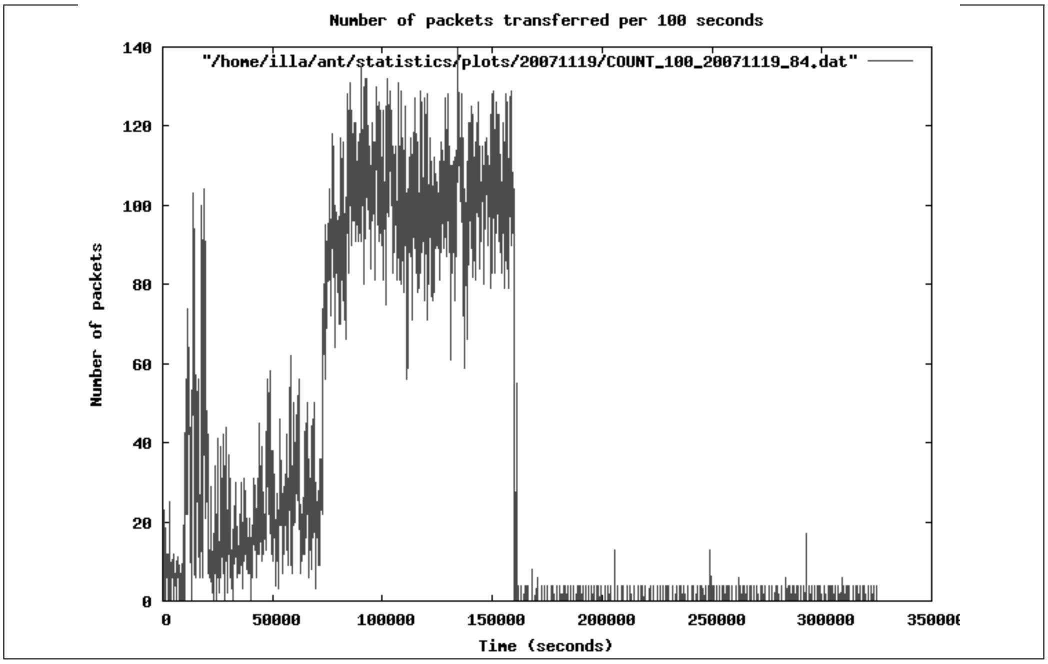


Figure 5.1: Packet count per 100 seconds

tries to connect the botmaster and afterwards, goes into the state of waiting for commands. The first spike at the 1000th second may be explained as either an update or the initial connection phase of the bot. Clearly, the most striking part of the graphic is the activity performed between 7000th and 160000th seconds. The activity is probably either a DDoS attack or a scanning performed to find new vulnerable machines, since the bots typically send several packets per second within DDoS attacks or scanning activities. After the activity is finished at the 160000th second, the graphic shows a regular activity that probably is the activity of exchanging $PING - PONG$ packetsd between the bot and the IRC server.

The Figure 5.2 shows the cumulative packet size change. The graphic is sketched with the intent of observing downloads. At about the 160000th second, there is a spike, which can not be observed from the Figure 5.1. The reason of the spike is a download that may be an update for the bot. The attack interval can also be recognized from the graphic, since after the 50000th second, there is a continuous block whose average packets size is bigger than the other blocks.

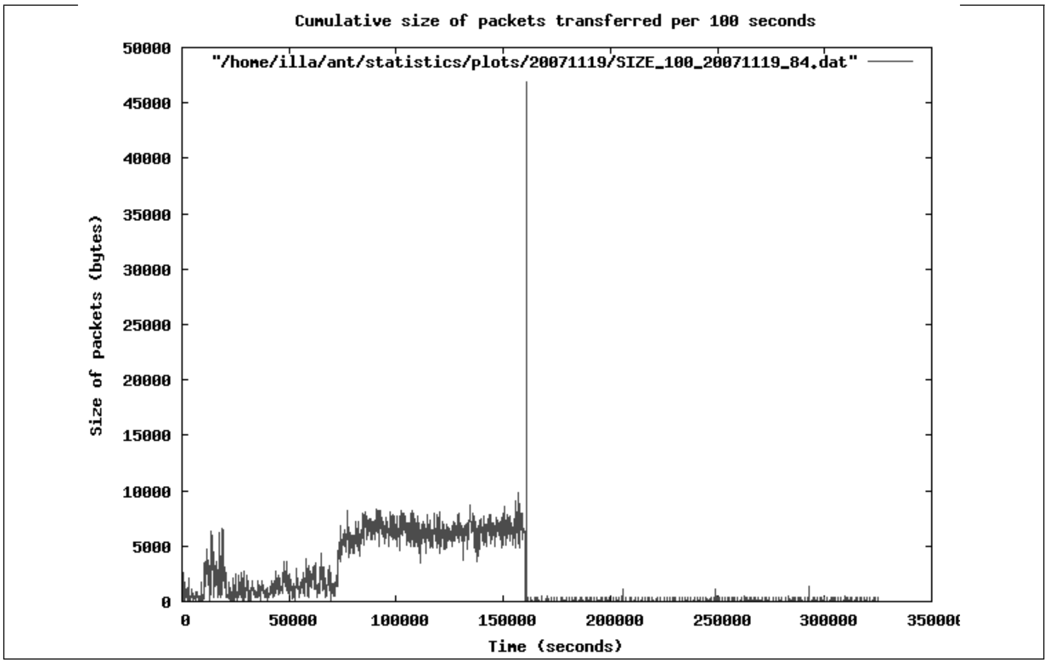


Figure 5.2: Cumulative packet size per 100 seconds

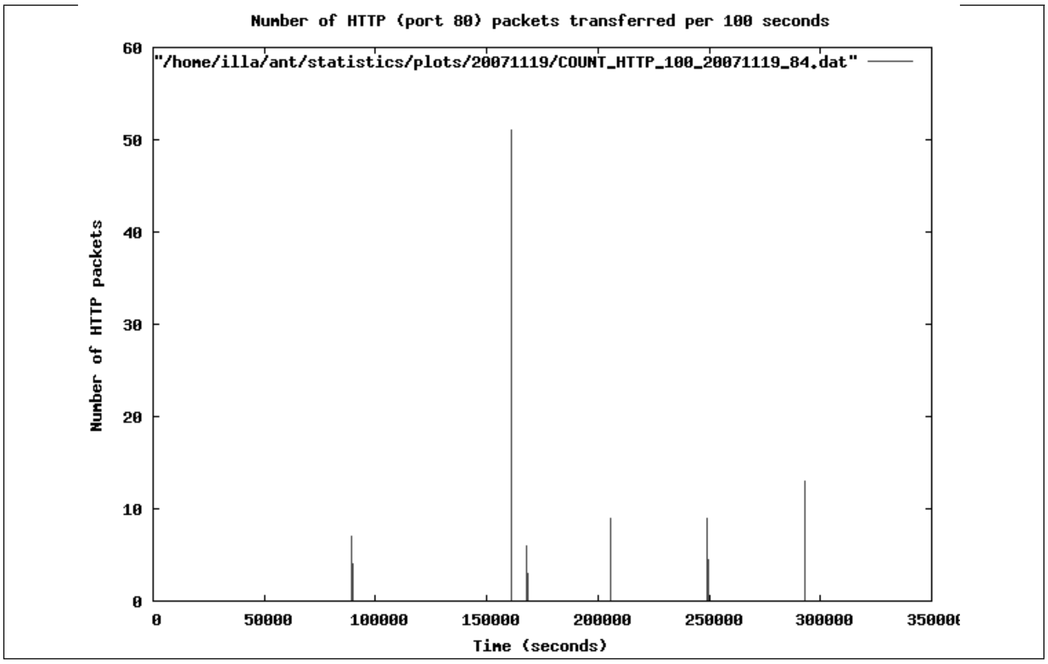


Figure 5.3: Count of HTTP packets per 100 seconds

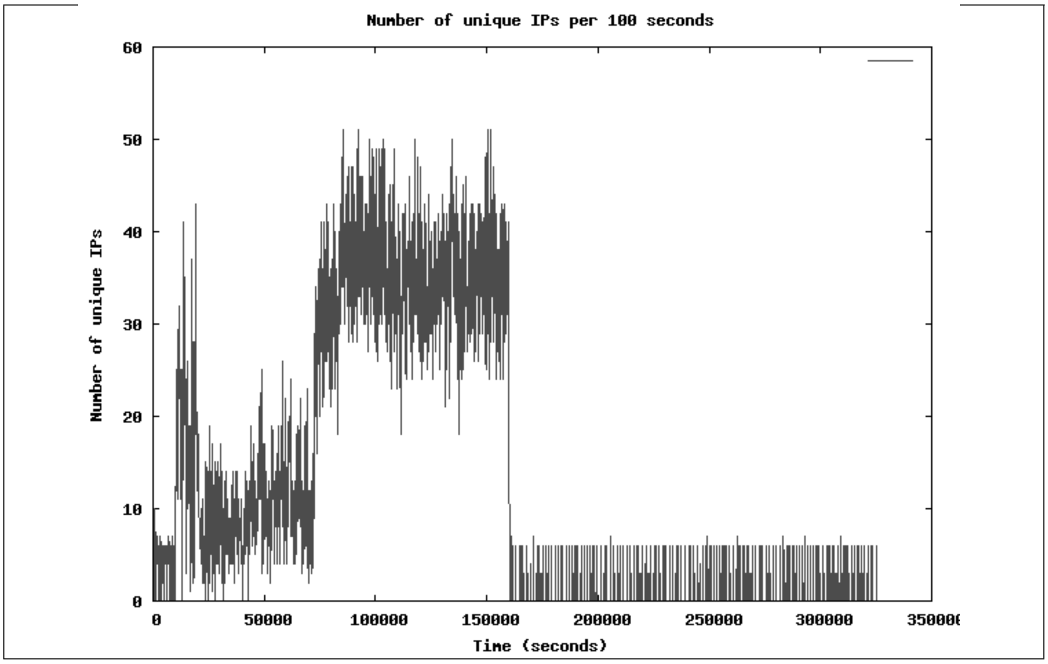


Figure 5.4: Count of unique IP addresses per 100 seconds

The Figure 5.3 is sketched either to observe the downloads that are done by the HTTP protocol or to differentiate IRC bots from HTTP bots. Since the bot is an IRC bot, it has not got frequent HTTP connections as it can be seen from the graphic. The spike that was observed in Figure 5.2 exists in the graphic that counts the HTTP packet as well. Thus, the download that is seen at about the 160000th second was done by using the HTTP protocol.

The graphics in the Figure 5.4 and Figure 5.5 are sketched to observe scans. There are two types of scanning mechanisms: the address scanning and the port scanning. Address scanning mechanism (a.k.a. horizontal scanning) scans a single port in a specified address space. The address scanning activity can be recognized by observing the graphic that counts packets with unique IP addresses, since starting connection attempts to different IP addresses will cause a spike in the graphic. Port scanning mechanism (a.k.a vertical scanning) scans a port range on a single IP address. Thus, because of the same reason that is described before, the port scanning activity can be observed from the graphic that count packets with unique port numbers. The scanning performed in the traffic capture is address scanning as it can be seen from the Figure 5.4. Obviously, we do not expect to see

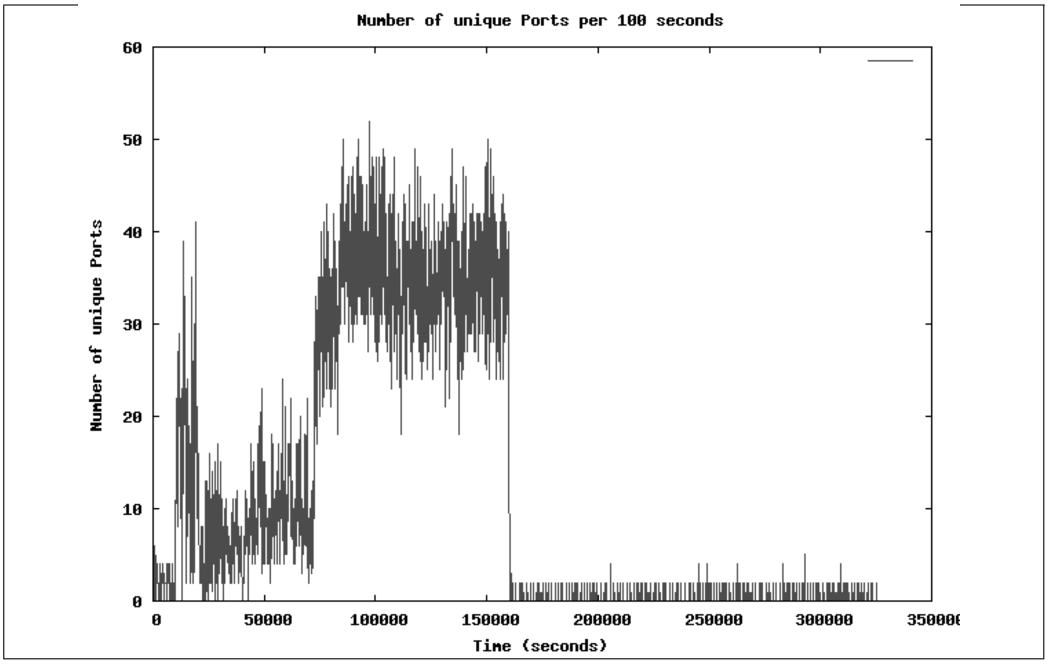


Figure 5.5: Count of port numbers per 100 seconds

the port scanning activity in the Figure 5.5, however two of the graphics show the same behavior, because of the weakness of the aggregate analysis. The aggregate analysis does not distinguish between the packets that are sourced by the bot and the victim machine. Thus, it also counts the unique port numbers that belong to the bot. Even though the bot scans an address range with the same destination port number, the source port number changes in every packet. The change at the source port number increases the count of unique port numbers. That is the reason of the similarity in the graphics.

The graphic in the Figure 5.6 shows number of non-ascii characters transformed in 100 seconds. The property of counting binary characters is used for making more precise decisions about downloads. While the file that is downloaded might be an executable, it might be a configuration file as well. Thus, the graphics are sketched in order to distinguish the executables from the normal files. After we analyze the Figure 5.6, we can accurately claim that the download at the 160000th second was an executable.

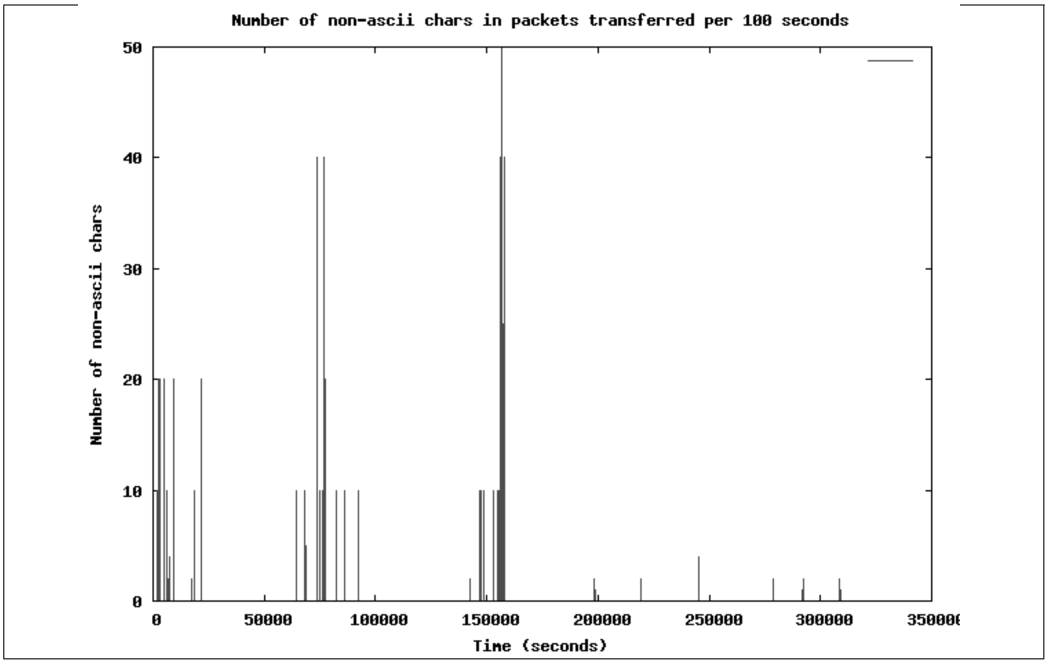


Figure 5.6: Number of non-ascii characters per 100 seconds

5.2 Connection Based Analysis

The connection based analysis examines the connections separately. The aim of the analysis is to get the information that cannot be obtained by the aggregate analysis. The aggregate analysis does not deal with different protocols in a generic manner. That is to say, in order to observe the activities performed by a specific protocol, such as HTTP, FTP, SMTP or UDP, we have to decide the port number that we want to analyze. Since, the botnets may use unknown protocols for their command and control; we cannot guess which port to listen without any pre-knowledge. Thus, the aggregate analysis is a restricted to analyze only the well-known protocols. Obviously, this property is its one of the major drawbacks. On the other hand, the connection based analysis has relatively more generic approach, since it produces statistics for each of the connection with a protocol independent manner.

In contrary to the aggregate analysis, the connection based analysis examines the traffic according to only two properties that are simply based on packet count and amount of the data per connection. The Figure 5.7 and Figure 5.8 show the

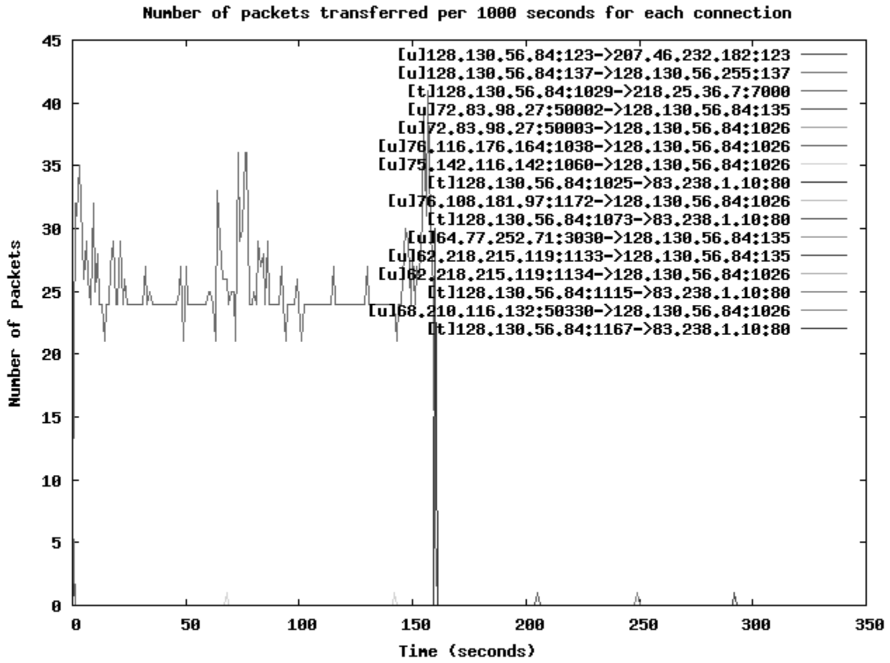


Figure 5.7: Count of Packets per connection

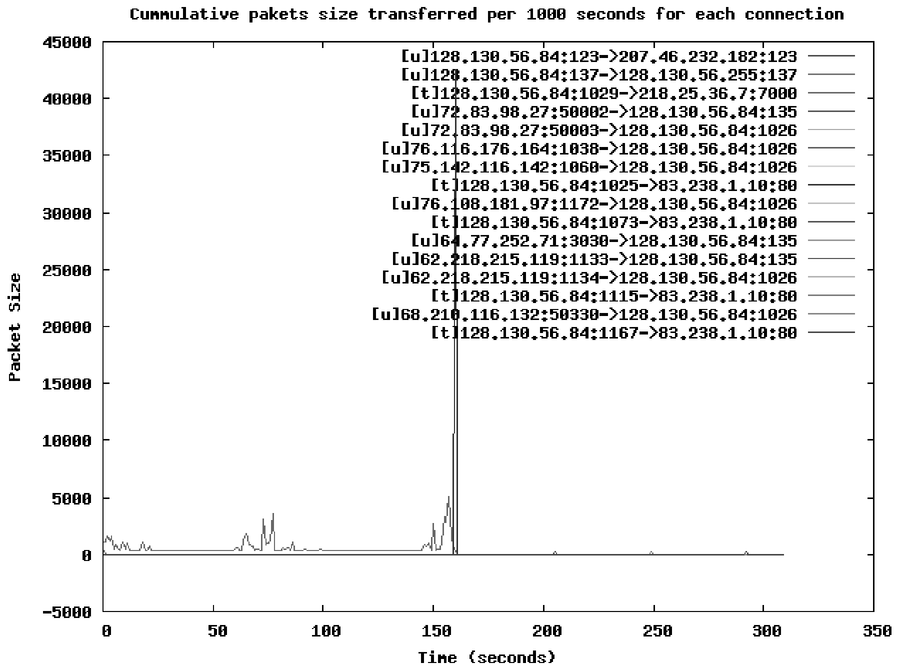


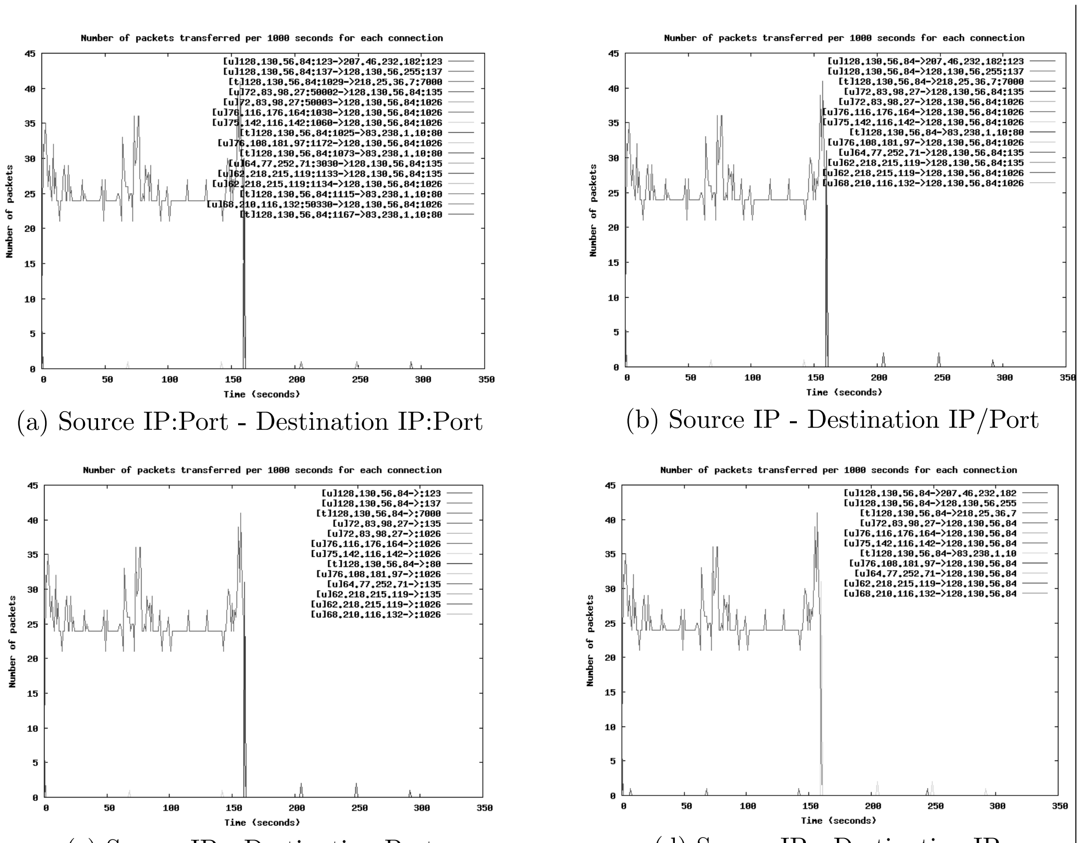
Figure 5.8: Amount of data flows per connection

graphics that are sketched for these properties. Since we are able to observe the attributes of each connection, such as the source and destination's IP addresses and port numbers, actually we can get all of the protocol specific information that we can get from aggregate analysis. Moreover, we have the opportunity to observe the status of other important unknown connections.

The connection based analysis filters half connections out and examines the remaining connections. Thus, the packets that belong to either scanning or DDoS attacks are not shown in the graphics. Since we can observe such attacks from the graphics that are sketched by the aggregate analysis, we choose to filter out them and focus on other important activities by doing connection based analysis. Connection based analysis rather aims to get more information about the connections that have the bot C&C and then, to find the connection that leads the bot to perform a specific attack.

Typically, the C&C communication is performed in a long term connection, since the bots connect to a channel and wait for command in an idle state unless a command is issued by the botmaster. This behavior can clearly can be seen from the graphic in the Figure 5.7. Probably, the long term connection that has the destination address as 218.25.36.7 : 7000 has the C&C communication. Another interesting thing that can be interpreted is the possible time interval that DDoS attack or scanning is performed. In the graphic the long term connection ends at about 160000th second and afterwards, any other long term connection does not exist. There are two possible reasons for that: first is that the bot finishes its network activities and second an attack, which consists of half connections, is started.

The graphic in the Figure 5.8 is mainly sketched in order to observe downloads in detail. The aggregate analysis concludes that at about the 160000th second, a download activity is performed. The connection based analysis not only approves that conclusion, but also shows information about the destination machine the file is downloaded from. As it can be seen from the graphic, the destination address is 83.238.1.10 : 80.



(a) Source IP:Port - Destination IP:Port

(b) Source IP - Destination IP/Port

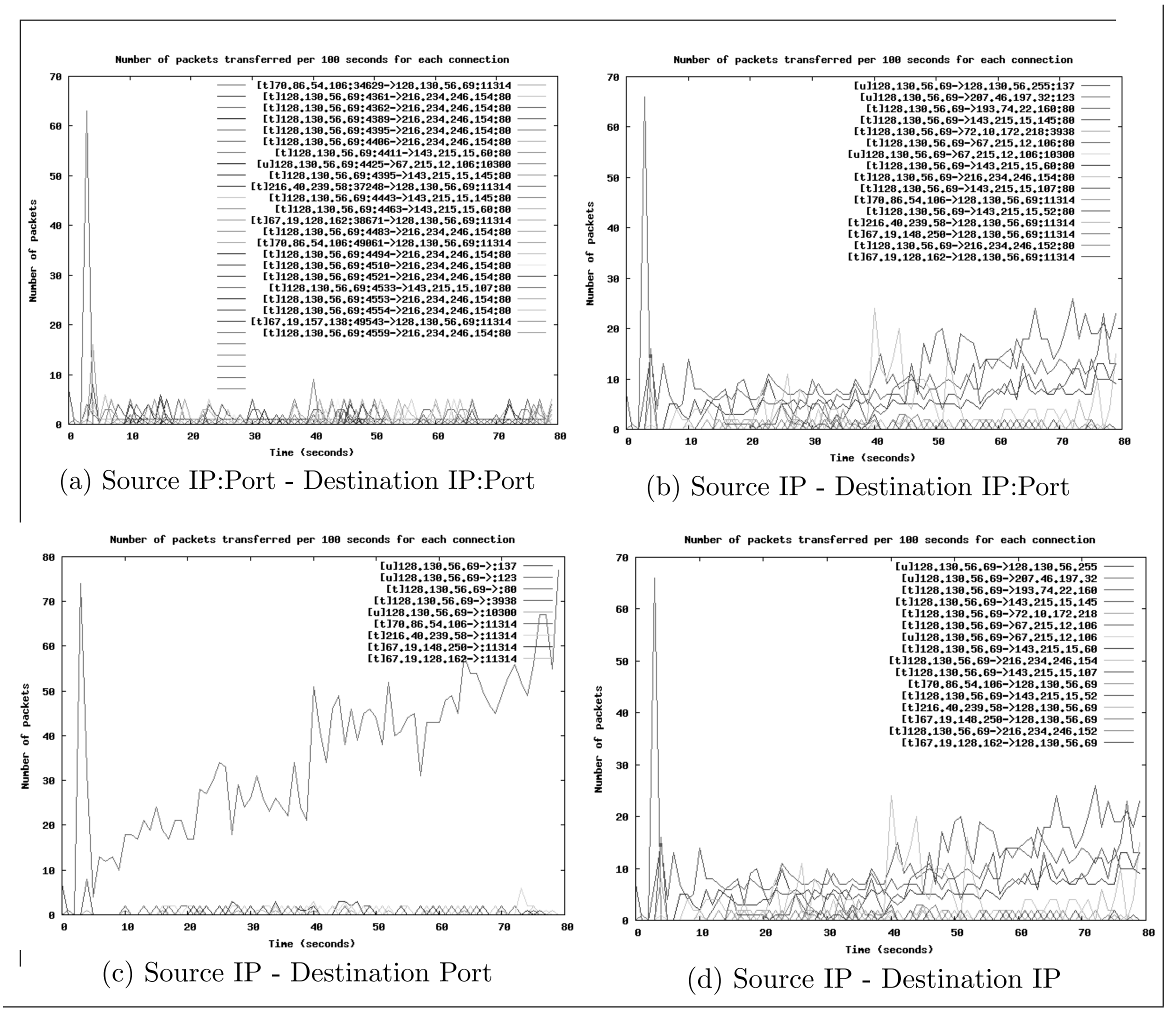
(c) Source IP - Destination Port

(d) Source IP - Destination IP

Figure 5.9: IP and Port Specific Connection Based Analysis of an IRC Bot

5.2.1 Detailed Connection Based Analysis Based on IP Addresses and Port Numbers

The connection based analysis gives good results for bot samples that produce relatively simple traffic traces. For example, the IRC bots generally do not have complicated network activities, since they only connect to one or more IRC channels or servers. Thus, there are a few connections in the graphic that allows us to interpret the behavior of the bots. Unfortunately, most of the bots produces complicated network traffics that consist of numerous connections, which make the graphic impossible to be interpreted as it can be seen from the Figure 5.10.



(a) Source IP:Port - Destination IP:Port

(b) Source IP - Destination IP:Port

(c) Source IP - Destination Port

(d) Source IP - Destination IP

Figure 5.10: IP and Port Specific Connection Based Analysis of an HTTP Bot
]

To simplify the graphics, we have made three more different analysis by modifying the connection tuple, which consists of source ip(SRC-IP), source port(SRC-P), destination ip(DST-IP) and destination port(DST-P). Thus, we have four different types of connection based analysis:

- *(SRC-IP,SRC-P,DST-IP,DST-P):* The connection consists of all of the members of a normal connection. Thus, the graphics analyze all of the connections separately. This analysis is applied to only IRC bots, since they generally have a few connections in their network traffic traces.
- *(SRC-IP,DST-IP,DST-P):* The analysis aims to find bots that have polling based C&C mechanisms. In the graphic, the connections that have the same destination address are shown in the same line.
- *(SRC-IP,DST-IP):* The analysis is done to observe frequently used destination IP addresses.
- *(SRC-IP,DST-P):* The analysis is done to observe frequently used destination port numbers.

We have sketched the graphics for two different bot samples by using all of the analysis strategies described above. Figure 5.9 shows network activities of an IRC bot. As it can be seen, the graphics that are sketched with the statistics that are produced with four of the approaches do not make a difference, since the IRC bots have a few long term connections and the response to the command issued by the botmaster in their traffic. On the other hand the Figure 5.10 shows the graphics for an HTTP bot sample. The first graphic is too complicated that nothing can be interpreted because all of the connections are handled separately. Obviously, other analysis schemes produce more promising graphics. The (SRC-IP,DST-IP,DST-P) analysis shows that the bot frequently connects to some different servers. The (SRC-IP,DST-P) analysis shows that the bot most frequently connects to the port number 80 and four outsiders frequently connect to the port 11314 on the machine the bot runs. And finally, the (SRC-IP,DST-IP) analysis shows the frequently connected hosts.

5.3 Observing Bots That Have Different Types of C&C Mechanisms

To date, there are three different types of C&C mechanisms: push style C&C, poll style C&C and P2P style C&C, which are are described in Section 2.3.2 in detail. In the following sections, we will give the conclusions that we get from the network analysis that are done for botnets with different types of C&C mechanisms.

5.3.1 Push Style Bots

The botnets that have push style C&C mechanisms have a central server to control bots. The bots connect to the central server and wait for the commands and this behavior constitutes the main characteristic of push style C&C. Thus, as soon as the bot is run in the compromised machine, it starts a network activity by trying to establish the connection to the central server. The network captures we have collected includes all of the traffic traces produced by the bot starting at the bot's run time. The Figure 5.1 proves that the bot really starts some network activities, as soon as it is run.

Since the main characteristic of push style bots is that the bots connect to a server and wait for the commands, obviously the traffic consist of long term connections and one of these connections have the C&C communication that might trigger the bot do an attack. The Figure 5.7 approves the existence of the main characteristic of push style bots.

5.3.2 Poll Style Bots

The botnets that have poll sytle C&C mechanisms have a centralized control over the bots as the botnets with push style C&C mechanisms have. However, the bots do not connect to a server and wait for the commands in an idle status.

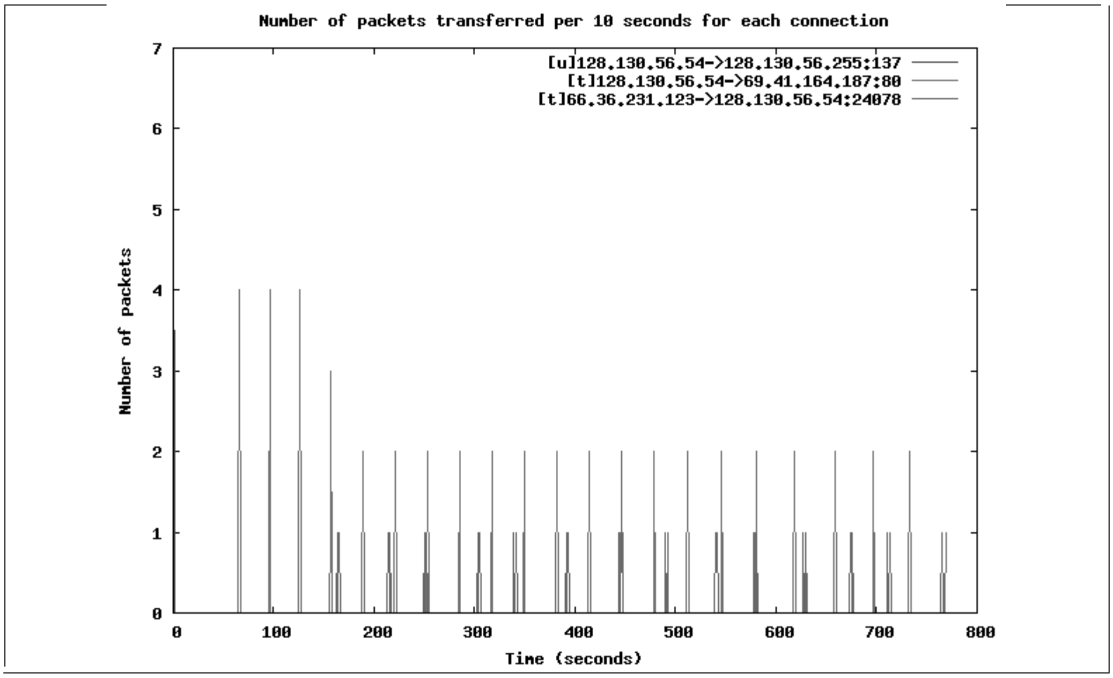


Figure 5.11: HTTP bots' characteristics

They poll the server for the commands. That is to say, they frequently connect to the server and check whether a command is issued or not.

The HTTP bots have a poll style C&C mechanism. They use the HTTP protocol to communicate with the botmaster. The botmaster installs a HTTP server and the bots frequently queries the server for the commands. The graphic in the Figure 5.11 shows the polling characteristic of HTTP bots.

5.3.3 P2P Bots

The P2P botnets do not have a centralized C&C mechanism; instead they have a decentralized C&C mechanism where the bots behave both as a server and a client. Thus to interpret the P2P bots' behaviors, neither the aggregate analysis nor the connection based analysis can be directly applied. The P2P bots that we have examined use a Kademlia based P2P C&C mechanism. When the bot sample is run in the compromised machine, it tries to join to the botnet. First, it creates a unique identity and declares itself to the botnet. Then, they exchange

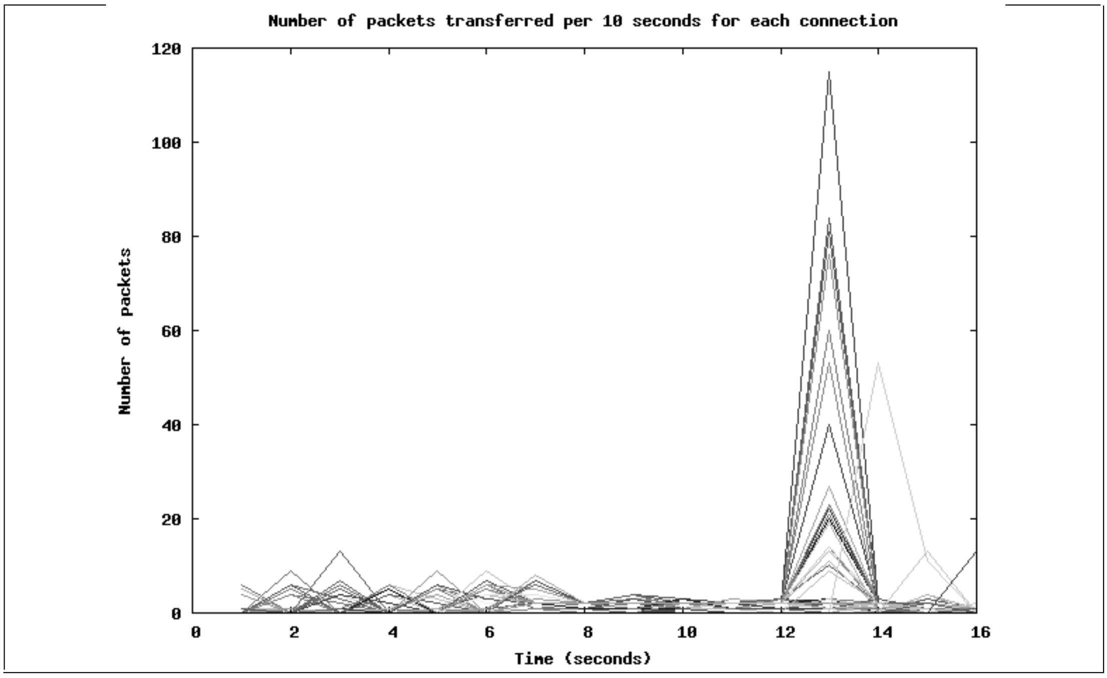


Figure 5.12: Count of packets produced by the storm sample for each connection

some information to construct the network to get or send the commands that are published by the botmaster. We could collect the network traffic just for about three minutes, because the P2P botnets produce an extreme amount of network traffic. Thus, the analysis interval for the graphics is 10 seconds.

We have analyzed Storm, which is the most well-known P2P today. The Figure 5.12 and Figure 5.13 shows the graphics produces with connection based analysis. As it can be seen, there are too many different connections that do not allow us to extract some information from the graphics. The only interesting thing is the sudden peak seen between 120th and 140th seconds. It may be the time interval where the command is published, since it is different than the regular activity of the bot.

The Figure 5.14 and Figure 5.15 shows the graphics that are sketched with aggregate analysis. Unfortunately, either the count of packets or the data transferred do not show any interesting information.

Generally, storm is known with its spamming attacks. The Figure 5.16 has

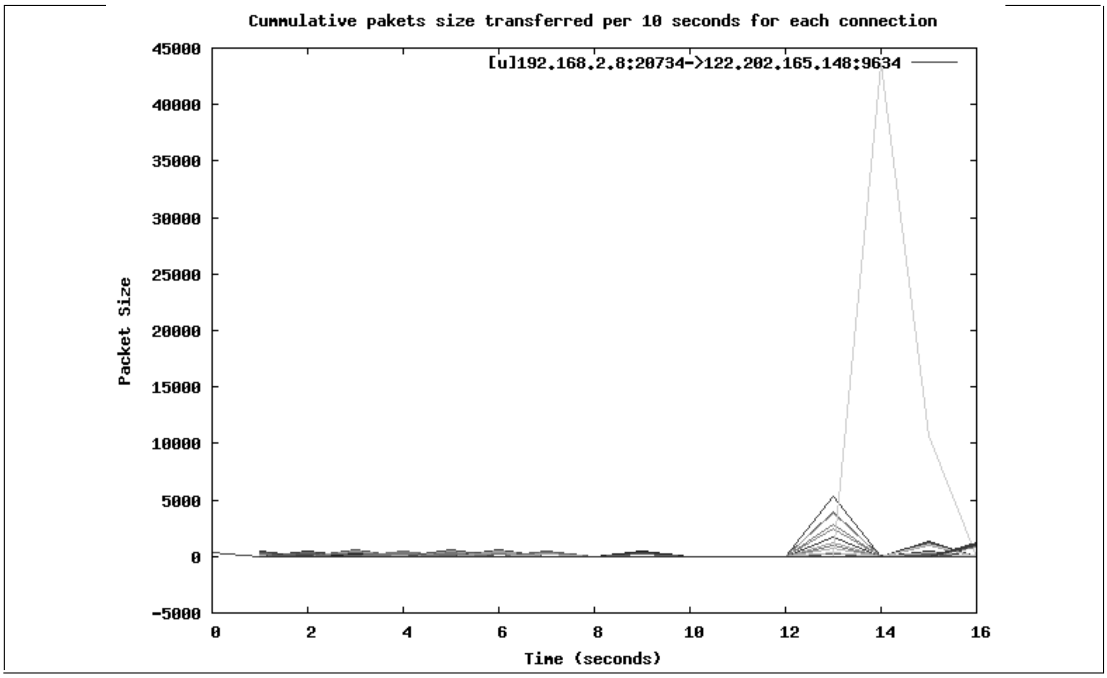


Figure 5.13: Amount of data transfered in each connection by the storm sample

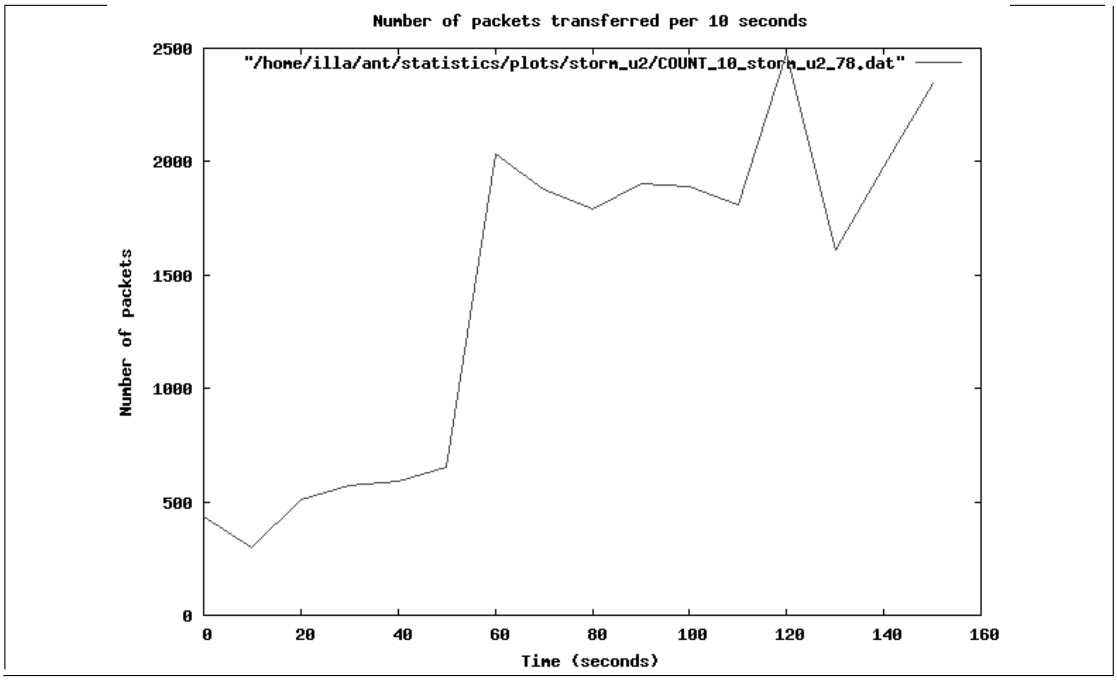


Figure 5.14: Total count of packets produced by the storm bot

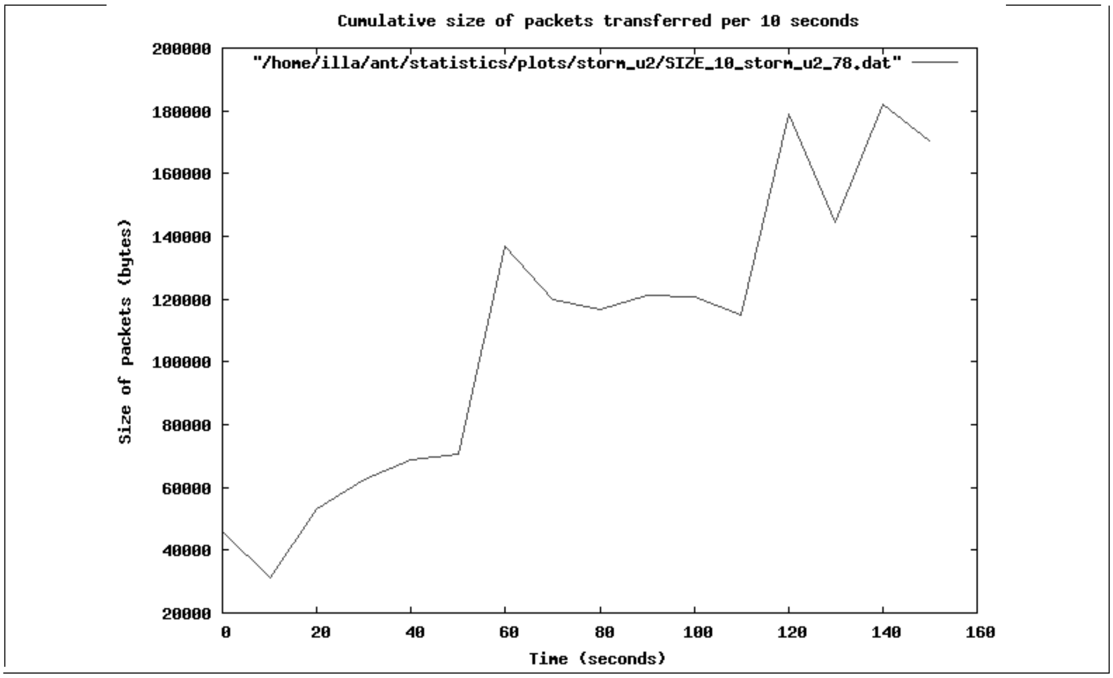


Figure 5.15: Cumulative amount of data produce by the storm bot

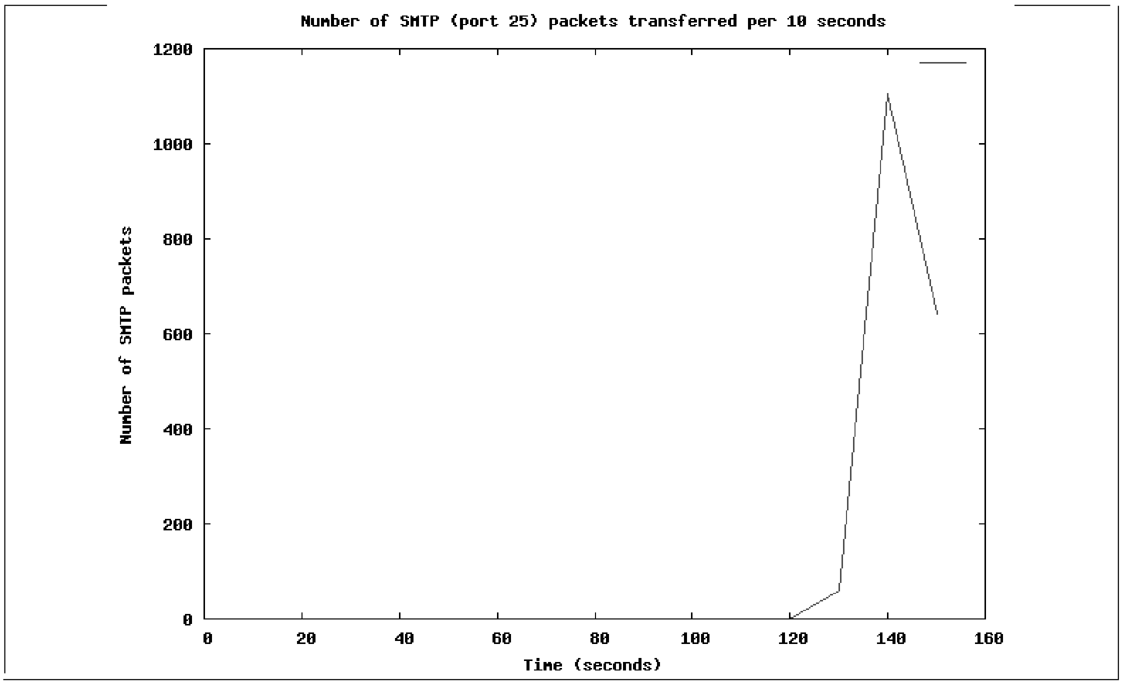


Figure 5.16: Count of SMTP packets produced by the storm bot

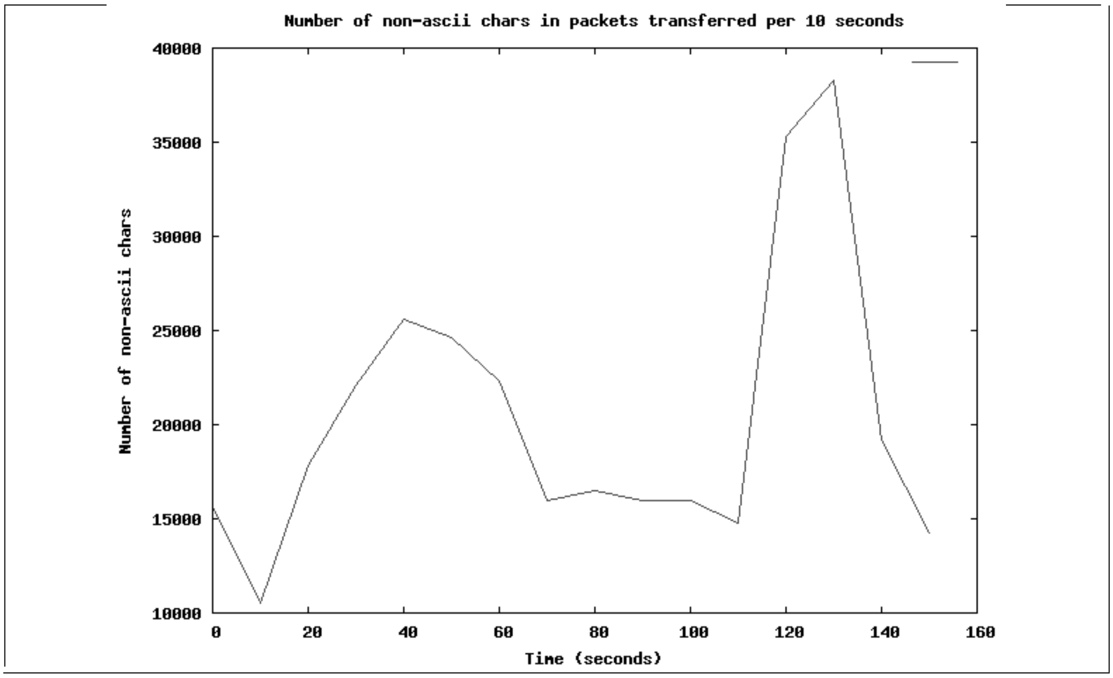


Figure 5.17: Count of non-ascii characters sent and received by the storm bot

the graphic that is sketched with the data produced by counting the smtp packets transferred. The graphic clearly shows that the bot starts the spamming activity at the 120th second.

The storm bot uses UDP for its C&C protocol that is Kademlia based as mentioned before. Thus, the C&C protocol is a binary protocol as the graphic in the Figure 5.17 shows. Unfortunately, this characteristic of storm bot makes it impossible to detect downloads by counting the non-ascii characters in the network traffic capture.

Chapter 6

Signature Generation

Traditional means of defense against bots rely on anti-virus software installed on user machines. Unfortunately, they have a limited detection scheme, which strictly depends on predefined signatures. Clearly, it is very difficult to update the signatures fast enough to keep up the speed of botnet evolution. The network administrators prefer to have a full control over the network they are responsible for. Installing anti-virus software or a malware scanner on the host machines prevents the administrator from observing the status of the computers, since host specific security gives the users independence. Unfortunately, there is always a possibility to have either careless or attacker users inside the network who makes the system vulnerable to bots or other malwares. Thus, the network administrators make use of intrusion detection systems (IDS) that can be installed on a vantage point at the network.

An IDS is responsible of monitoring either the inbound or the outbound traffic that flows on the machine it is installed on. While it is monitoring the traffic, it searches for known signatures that correspond to malicious traffic. If a malicious content tries to pass through the network, the IDS may either raise an alarm to inform the administrator or drop the connection to prevent the malicious traffic reach the victim.

To date, there are two well-known intrusion detection systems: Bro [24] and

Snort [28]. While Snort uses only content-based signatures to detect the malware, Bro uses both content-based and network-based signatures. Network based signatures are also called as behavioral signatures, since they are used for monitoring network behaviors of the machines. Network-based signatures are typically designed for detecting scans or denial of service attacks that may either be sourced by a local machine or a remote machine. Clearly, the content-based signatures are used for matching known patterns that come within the transferred packets. The signature matching techniques applied by Snort and Bro are:

- String matching at arbitrary payload offsets
- String matching at fixed payload offsets
- Matching regular expressions

The goal of our system is to create detection models for various bot families, as described in Chapter 3. The detection models involve generating content-based signatures that can be deployed to IDS systems such as Bro [24] and Snort [28]. Normally, the content-based signatures are generated with manual effort. The malware analysts examine malware samples and try to find some common patterns that can identify malware. A number of systems [14, 31, 20, 22] have been proposed that are able to automatically generate signatures to detect worms. The worm signatures consist of some tokens that are extracted from the worm executable itself. Contrary to signatures for other malware, signatures to detect bots do not consist of bot executable, but the commands that are issued by the botmaster. To date, an automated signature generation scheme for bots does not exist. Thus, with our system, we aim to fill this gap.

6.1 Signature Quality

Ideally, a signature generated for a specific command that belongs to a bot family should detect all of the members of the family as soon as the specific command is issued. Moreover, the signature must not match other traffic patterns that do not belong to the communication between the bot and the botmaster. The quality of the signatures is evaluated with two concepts: *sensitivity* and *specificity*.

- *Sensitivity:* The sensitivity of a signature is related to the rate of the *true positives*. In traffic that consists of both bot C&C and normal traffic, the fraction of bot C&C flows matched constitutes the *true positives*. Sensitivity is reported as $t \in [0, 1]$, the fraction of true positives. Clearly, a sensitive signature has approximately $t \simeq 1$.
- *Specificity:* The specificity of a signature is related to the rate of the *false positives*. In a mixed traffic, the fraction of none bot C&C flows matched constitutes the *false positives*. Specificity is reported as $(1 - f) \in [0, 1]$, where f is the fraction of false positives. The ideal signatures have to be specific enough that they do not produce any false positives.

In practice, it is impossible to have perfect sensitivity and perfect specificity together, since these two concepts have a tension between themselves. That is to say, when a signature generation scheme tries to achieve the perfect sensitivity, it looses the perfect specificity. Thus, the signature generation scheme tries to have the balance between sensitivity and specificity by finding the most reasonable values for each.

6.2 Content-Based Signatures

We have analyzed various kinds of content-based signature generation schemes [14, 31, 20, 22] and decided to generate token subsequence signatures since they can also be deployed as regular expressions to known intrusion detection systems, such as Bro and Snort. In the flowing section, we briefly explain the content-based signature generation schemes to date.

6.2.1 Substring Signatures

Honeycomb [17], Autograph [14] and EarlyBird [31] provide automatically generated pattern-based signatures that are used for detecting worms. The signatures are single substring signatures that are produced from worm executables. Unfortunately, single substring signatures have very high false positives rate unless the signature is long enough. Since, it is difficult to ensure the existence of such a long substring; the single substring signatures are not specific enough.

Since the single substring signatures are not specific enough to identify worms, which mostly consists of binary data that rarely have commonalities with benign traffic; they are too general for signatures that must consist of bot C&C as well. Typically, IRC bots use clear-text C&C and the longest common substrings found are generally some strings that are likely seen in the normal web traffic. For example, `advscan` command basically triggers scanning activity on bots. Supposing that the longest common substring produced is `scan`, the signature clearly will trigger so many false positives. We do not prefer to use the single substring signatures, because we want the quality of the signature to be high enough that it does not generate so many false alarms that may disturb the network administrators.

6.2.2 Conjunction Signatures

Conjunction signatures are one of the signature generation schemes provided by Polygraph [22]. The signatures consist of a set of tokens and they are matched only if all of the tokens in the set are found in the packet, in any order.

6.2.3 Token Subsequence Signatures

Token subsequence signatures consist of a set of tokens as conjunction signatures. However, the token subsequence signatures have an ordered set of tokens. The signature is matched only if all of the tokens in the set are found in an ordered fashion. Obviously, the quality of token subsequence signatures are better, since the necessity to match only tokens in order decreases the count of false positives.

The most important property of the token subsequence signatures is that the signatures can be deployed as regular expressions to the current intrusion detection systems without any effort. We have used token subsequence signature scheme to generate signatures for bot C&C, because of two reasons: first is that they can be expressed easily as regular expressions and second is that the signatures are very specific to bot C&C that do not trigger so many false positives.

6.2.4 Bayes Signatures

Bayes signatures consist of a set of tokens as well as the token subsequence and conjunction signatures. Each of the tokens is associated with a score and an overall threshold. In contrast with the exact matching offered by the other schemes, bayes signatures provide a probabilistic matching. For each of the tokens a probability score is computed and only if the score is over a threshold, the token is added to the signature.

6.3 Signature Generation Algorithms

As it is mentioned before, we have preferred the token subsequence signatures because they can be expressed as regular expressions and the count of false positives they generate are relatively low. To generate token subsequence signatures, a couple of off-the-shelf algorithms can be used. In the following sections, we describe some algorithms we have made use of.

6.3.1 Longest Common Substring Algorithm

The longest common substring problem is to find the longest string that is a substring of all of the strings in a set. The algorithm can be generalized to find more than one common substring from the set. As a preprocessing step before signature generation, we extract common tokens that are restricted to have a minimum length. The reason of performing a preprocessing step is to speed up the signature generation step by taking the useless tokens out. To this end, the generalized longest common subsequence algorithm (a.k.a k-common substring algorithm) finds all of the tokens with a minimum length λ that occur in at least κ out of the total n snippets of a behavioral cluster.

The longest common substring algorithm can be solved with three different approaches: first by making use of generalized suffix trees, second by making use of suffix arrays and the last by dynamic programming. We have evaluated all of the algorithms and experiments showed that suffix trees and suffix arrays are more effective and faster than the dynamic programming algorithm. Thus, we have implemented the system by using suffix trees and arrays not by using dynamic programming approach.

6.3.1.1 Suffix Trees

A *suffix tree* (a.k.a PAT tree) is a data structure that positions all of the suffixes of a given string in a way that allows for a particularly fast implementation of many important string operations. Constructing the tree for a string takes time and space linear in the length of the string. Although the suffix tree of a string needs more space than the space needed for the string itself, the speedup it provides makes the algorithm reasonable to be used for several purposes.

A *generalized suffix tree* is a suffix tree that contains a set of strings. The most important functionality of generalized suffix trees is that it can be used to solve longest commons subsequence problem for a set of strings. The problem is solved by locating all of prefixes of the strings in one suffix tree and then, finding the deepest internal nodes which has leaf nodes from all the strings in the subtree below it. The Figure 6.1 shows the generalized suffix tree constructed for two strings: `abbab` and `aabab`. To differentiate the strings the strings are padded with unique string terminators, such as $1 or $2. As it can be seen from the Figure 6.1, the longest common substring is the characters on the path whose

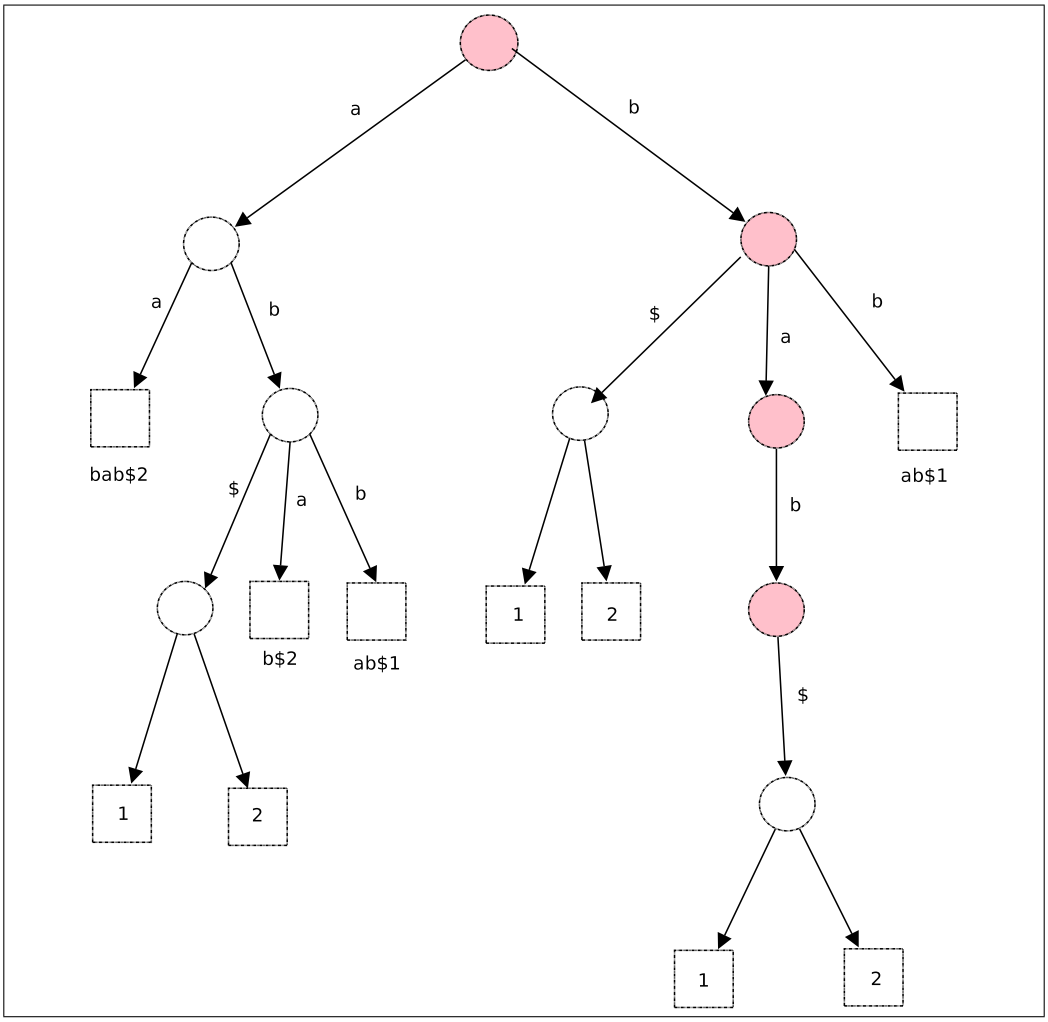


Figure 6.1: Generalized Suffix Tree Representation of two strings, abbab and aabab

leaf node has both 1 and 2 and is the deepest in the tree. The longest common substring of the strings abbab and aabab is bab that is on the path with filled nodes.

We have made use of a generic suffix tree library, libstree [16] that is written in C programming language. The library can handle different types of data structures as elements of the string that may consist of also non-ascii characters. The suffix tree generation in the library is implemented using the algorithm by Ukkonen. Since the libstree can handle multiple strings per the suffix tree, it may be used to find longest common substrings, which we need.

6.3.1.2 Suffix Arrays

The suffixes
a b a c d a c b b
a c b b
a c d a c b b
b
b a c d a c b b
b b
c b b
c d a c b b
d a c b b

Table 6.1: The suffix array constructed for the string abacdacbb

To find the signatures, we have made use of some piece of code already exists in Polygraph [22], which uses suffix trees to extract the tokens. After polygraph was published, Li et. al. published Hamsa [20] which also aims to generate signatures for polymorphic worms. In their paper, the authors claim that by using suffix arrays instead of suffix trees, the speed of generating the signature significantly increases. Thus, we have decided to implement the signature generation code also by using suffix arrays to see if they are actually right.

A suffix array is an array giving the suffixes of a string in lexicographical order. For example, nine characters long string abacdacbb has nine suffixes and the suffix array constructed for the string is shown in the Table 6.1. Suffix array of a string can be used as an index to quickly locate all possible substrings within the string. Since the substrings are sorted in lexicographical order, by using the binary search easily the substring can be found.

The suffix arrays can also be used for finding the longest common substrings between two strings, since searching for arbitrary strings is very fast with suffix arrays. After we implemented the longest common substrings problem with suffix arrays, we have seen that it increases the performance as authors of Hamsa claim.

6.3.2 Longest Common Subsequence Algorithm

The longest common subsequence problem is finding the longest subsequence that exists in all of the strings in a set. Thus, the tokens sequence signatures for the bot C&C can be generated by making use of the longest common subsequence algorithm. A subsequence of two strings is an ordered sequence of bytes that do not need to be contiguous. For example, the longest common subsequence of two strings *xxxsmp1xxxxsmp2xxx* and *ysmp1yyyysmp2yy* is *smp1smp2*.

The typical solution to longest common subsequence algorithm is done by dynamic programming and the solution for two strings $X_{1..n}$ and $Y_{1..m}$ is given as:

$$LCS(X_{1..i},Y_{1..j})=\begin{cases} \phi & if\, i=0\, or\, j=0 \\ LCS(X_{1..i-1},Y_{1..j-1})+x_i & if x_i=y_j \\ LCS(max(LCS(X_{1..i},Y_{1..j-1}),LCS(X_{1..i-1},Y_{1..j}))) & else \end{cases} \tag{6.1}$$

6.4 Generating Signatures for Detecting the Bots

Signature generation phase is the last step of our system, which firstly analyzes the bot network captures according to properties described in Section 5.1 to specify the behavioral changes. We make the assumption that the bot command that triggers the behavioral change is issued maximum in 100 seconds before the behavioral change is recognized. Thus, the signature generation concentrates on the network capture generated between the time of behavioral change and 100 seconds before it. We call that 100 seconds network capture preceding the behavioral change as network traffic snippet.

The input to the signature generation algorithm is a set of clustered network traffic snippets that are associated with specific behavioral profiles. The behavioral profile of such a cluster represents a bot activity, such as scanning, e-mail spamming or one of the other malicious activities that are explained in Section 2.2. We assume that all of the botmasters who use the C&C mechanism of a specific bot family should use the same commands. Thus, all of the bots that belong to the same family should behave the same when they get the same command. That is to say, the network traffic snippets that have the same behavioral profile should include the commands that are similar to others in the cluster. These similar commands constitute the token signatures that are used to detect bot C&C communication.

Hereinafter, when we mention about a set of snippets, it must be understood that the set has snippets that have the same behavioral profiles. Because we cannot locate the exact time where the command is issued, we examine all of the data transferred in 100 seconds interval. Obviously, the data consists of a mixture of network traffic with different packets belong to different protocols. The signature generation algorithm is responsible for arranging the snippets such that those are put together in a cluster that likely contain the same command.

As mentioned in Section 6.2, to extract common substrings from network snippets that belongs to a behavioral cluster, we leverage the existing signature generation techniques. We experimented with different signature generation schemes and decided to stay with token subsequence signatures as produced by Polygraph [22].

Token extraction is performed in two phases: First, we cluster the payloads in the snippets according to their similarities, and second, we find the longest common subsequence in each cluster separately. We leverage the agglomerative hierarchical clustering algorithm implemented in Polygraph, with several modifications. Polygraph computes a score that is dependent on the common tokens' false positive rate, which is defined as the token sequence's matching rate on an innocuous pool. As there is no innocuous pool in our setup, we define the similarity score S between two payloads, P_1 and P_2, that have n common tokens $t_1, \ldots, t_n$, as follows:

$$S = \sum_{i=1}^{n} \left(\frac{len(t_i)}{len(P_1)} + \frac{len(t_i)}{len(P_2)} \right) /2 \tag{6.2}$$

The clustering phase starts after we put each payload into a separate cluster. Then, similar clusters are successively merged into larger clusters. Two clusters are merged only if the payloads in the clusters have a similarity score greater than a minimum threshold that is empirically chosen to prevent the system from aggressively removing tokens from the signatures, as this leads to over-generalization and the loss of relevant information.

Basically, the clustering is done by the following steps;

1 Creating separate clusters for each of the payloads that are transferred in the snippets that belongs to a specific behavioral cluster.

2 Extracting common tokens for each possible pair in the input set and at the same time, calculating the score from the extracted common tokens.

3 Sorting the pairs according to their scores to find a starting point for merging.

4 As long as there are unmerged pairs,

1 Choosing the pair with highest score, merging the clusters that own the payloads of the pair and removing the pair from the unmerged pairs list

2 Searching other unmerged pairs to find the pairs that have one of the members of the recently merged pair. If it is found and the other payload does not decrease the score of the cluster to be under the threshold, when it is also merged to the cluster, merging it too

3 Sorting unmerged pairs

The algorithm above chooses the clusters to be merged with a greedy method. Since the greedy approach may reach local minimum instead of global minimum, the noise in the snippets may prevent producing ideal signatures. Such problems generally occur while generating signatures for IRC bots. As it is known, IRC servers send long topic packets when a client connects. The topic packets may consist of the name of the server, the name of the channel, list of the connected user names and some advertisement. Coincidentally, the long topics packets may have highest similarity score with the packets that have the command. Obviously, this coincidental similarity may lead to lose the command substring and produce signatures that consist of just channel properties. Nevertheless, we have used the greedy method and most of the time generated signatures have reasonably good quality.

The final step of the signature generation is to generate the token subsequence signatures. After the clustering of the payloads is finished, we have some clusters that have similar payloads. To find the token subsequence signatures, we find the longest common subsequences in each cluster separately.

Chapter 7

Evalution

The purpose of the evaluation is to demonstrate that our system can generate content-based signatures that are capable of detecting bot-infected machines with a low false positive rate. To this end, we analyzed a sample set of 283 bots. 251 of the samples are obtained via Anubis and afterwards, executed in our traffic environment over a period of 43 days. The remaining 32 traffic captures include Storm traffic and they are generated seperately at the University of Mannheim.

Bot family	#Models	#Token sequences
IRC1	2	12
IRC2	2	7
IRC3	2	10
IRC4	4	14
IRC5	7	31
IRC6	2	5
HTTP	2	6
STORM	1	11
TOTAL	22	96

Table 7.1: Numbers of detection models and total numbers of token sequences generated for each bot family.

Using the 258 samples, which are clustered into 6 IRC, 1 HTTP and 1 Storm families, we have produced 22 behavioral profiles. Table 7.1 shows the number of behavioral profiles generated for each bot family and the number of content-based signatures that identify each behavioral profile.

Figure A.1 shows one of the content-based signatures that identifes a behavioral profile for an IRC bot family. This signature consists of 7 token sequences that are generated from the network snippets which have the traffic captured before the response activity. Generally, only one of the token sequences has the command issued by the botmaster. For example, the sixth token sequence in the Figure A.2 has the `.asc` command and some parameters for it.

The signatures in the Figure A.6 and Figure A.7 prove that our system can not only produce signatures for IRC bots, but also the HTTP bots. Since the HTTP bots use poll-style C&C mechanism, the bot can get the command only if it demands it from the botmaster. Typically, the bots send some information about the compromised machine by passing some parameters to the command server. The signatures that are produced from the outbound traffic may also identify the behavior. Thus, the signatures for HTTP bots are generated for both inbound and outbound traffic.

Finally, we could also produce a signature for storm bots. Normally, producing signatures for Strom bots is a challenging task, because the botmaster does not use a central server to send the commands and the commands sent are XOR encrypted. Thus, it is impossible to generate signatures that identify a Storm command. Nevertheless, we have analyzed the output produced for Strom bots. Even though the signatures do not include tokens that are a part of the attack command, they include some tokens that shows the command is issued as it can be seen from the Figure A.8.

7.1 Signature Quality

In order to understand whether the signatures we produce for each family are useful or not, we evaluated the quality of them in two steps. In the first step, we compared our signatures with human-generated signatures, which were written by human experts. Of course, we expect to find significant similarity between them.

As it can be clearly seen from the signatures generated in Appendix A, we have three types of signatures:

- *Very specific signatures:* The token sequences contain substrings of a specific botnet's communication protocol. Clearly, these signatures will likely not cause false positives. However, they may only detect bots that belong to a specific botnet, not the other bots that are a member of the same bot family.
- *Very generic signatures:* The token sequences contain substrings that are related to the communication protocol that is made use of for C&C. For example, if the botnet uses IRC protocol for C&C, the signatures may contain only substrings that are used also in benign IRC communication. Obviously, this type of token sequences cause high false positive rate.
- *Command signatures:* The token sequences contain substring that are a part of the command issued by the botmaster.

The signatures that we have generated for each behavioral profile have at least one token sequence that has the command that we are looking for. We have compared the command signatures with the human-generated signatures that are deployed to Snort. All of our token sequences that fall into the third category are nearly the same as the Snort signatures. For example,the corresponding Snort signatures of `(’@admin.’, ’ PRIVMSG #’, ’# :.advscan ’, ’ -’, ’ n’)` is |2E|advscan|20.t

As a second step, we analyze the capability of our system to detect novel binaries. We have deployed our signatures to a Bro sensor in front of several malware analysis machines. None of the malware samples that are executed in the analysis machines are the bot samples we have executed in our test environment. In total approximetaly 800 malwares are executed and our system detected 19 bots. Thus, our system is capable of detecting bot-infected machines.

7.2 Real World Deployement

In order to evaluate our signatures to find the false positive rate, we have deployed them on a Bro sensor which is in front of a university network which consists of several student computers. During a testing period of 24 hours, the sensor monitored millions of connections. In total, the signatures triggered 402 times, with five different signatures matching on the network traffic. We have observed that some of the token sequences that fall in to the second category(*very generic signatures*) matched the benign traffic. Since for our system is not enough only to match the content-based signatures, we did not alert such matches. Our system alerts only if the response behavior is also observed after the content-based signatures is detected. Our system raised only one false alert, because by chance our signatures matched the traffic produced by eDonkey file sharing program and since the file sharing programs produce dense traffic, we recognized it as a scanning. Nevertheless, one false positive is acceptable especially in such a large network. We can conclude that our signatures can detect bot-infected machines with a low false positive rate.

Chapter 8

Conclusion

Today, botnets constitute a big treat against Internet users. They perform malicious activities that aim to steal important secret information, obstruct working of a system, make advertisements or send junk e-mails, etc. In this thesis, we present a system that monitors network traffic in order to examine it for signs that indicate the presence of bot-infected machines. To this end, we propose a detection model that focuses on the most distinguishing characteristic of bots that they receive commands and carry out some corresponding activities. In order to create our detection models, we observe the network traffic. Based on these observations we generate content-based and network-based signatures that can be deployed to known intrusion detection systems. Our approached relies on neither the bot spreading vector nor specific bot properties such as the command and control protocol. Thus, we were able to generate signatures for various bot families that use different command and control protocols, IRC, HTTP or P2P. Finally, the experiments showed that our system is capable of detecting bot-infected machines with a low false positive rate.

Bibliography

[1] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. C. Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *9th Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 165–184, 2006.

[2] E. Balas and C. Viecco. Towards a Third Generation Data Capture Architecture for Honeynets. In *6th IEEE Information Assurance Workshop*. West Point, 2005.

[3] U. Bayer. Anubis: Analyzing Unknown Binaries. `http://analysis.seclab.tuwien.ac.at/`.

[4] M. Christodorescu and S. Jha. Testing Malware Detectors. In *ACM International Symposium on Software Testing and Analysis (ISSTA)*, 2004.

[5] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant. Semantics-Aware Malware Detection. In *IEEE Symposium on Security and Privacy (Oakland)*, 2005.

[6] D. Dagon, G. Gu, C. Lee, and W. Lee. A Taxonomy of Botnet Structures. In *Annual Computer Security Applications Conference (ACSAC)*, 2007.

[7] M. de Hoon, S. Imoto, J. Nolan, and S. Miyano. Open Source Clustering Software. *Bioinformatics*, 20(9), 2004.

[8] D. Dietrich. Distributed Denial of Service(DDoS) Attacks/tools. `http://staff.washington.edu/dittrich/misc/ddos/`, 2005.

[9] T. F. S. Foundation. the GNU Compiler Collection. http://gcc.gnu.org.

[10] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *16th Usenix Security Symposium*, 2007.

[11] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008.

[12] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and Mitigation of Peer-to-Peer based Botnets: A Case Study on Storm Worm. In *First Usenix Workshop on Large-Scale Exploits and Emergent Threats(LEET'08)*, 2008.

[13] C. Kalt. Internet relay chat: Architecture. RFC2810.

[14] H. Kim and B. Karp. Autograph: Toward Automated, Distributed Worm Signature Detection. In *13th USENIX Security Symposium*, pages 271–286, August 2004.

[15] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based Spyware Detection. In *15th Usenix Security Symposium*, 2006.

[16] C. Kreibich. libstree-A generic suffix tree library. http://www.icir.org/christian/libstree/.

[17] C. Kreiblich and J. Crowcroft. Honeycomb - creating intrusion detection signatures using honeypots. In *the Scond Workshop on Hot Topics in Networks(HotNets-II)*, November 2003.

[18] C. Kruegel, W. Robertson, and G. Vigna. Detecting Kernel-Level Rootkits Through Binary Analysis. In *Annual Computer Security Applications Conference (ACSAC)*, 2004.

[19] J. Leyden. Rise of the Botnets. http://www.theregister.co.uk/2004/09/20/rise_of_the_botnets/, 2004.

[20] Z. Li, M. Sanghi, Y. Chen, M. Kao, and B. Chavez. Hamsa: Fast Signature Generation for Zero-day Polymorphic Worms with Provable Attack Resilience. In *IEEE Symposium on Security and Privacy*, 2006.

[21] P. Maymounkov and D. Mazieres. Kademlia: A Peer-to-peer information system based on the XOR metric. In *1st Workshop on Peer-to-Peer Systems(IPTPS)*, March 2002.

[22] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *IEEE Symposium on Security and Privacy*, pages 226–241, 2005.

[23] Norman. Norman Sandbox: Malware Analyzer. `http://www.norman.com/microsites/nsic/`.

[24] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31, 1999.

[25] G. Popek and R. Goldberg. Formal Requirements for Virtualizable Third Generation Architectures. *Communications of the ACM*, 17(7):412 – 421, July 1974.

[26] N. Provos. A Virtual Honeypot Framework. In *13th USENIX Security Symposium*, pages 1–14, 2004.

[27] Qemu. Qemu Open Source Processor Emulator. `http://fabrice.bellard.free.fr/qemu/`, 2008.

[28] M. Roesch. Snort - Lightweight Intrusion Detection for Networks. In *13th Systems Administration Conference (LISA)*, 1999.

[29] J. Rutkowska. Red Pill... or how to detect VMM using (almost) one CPU instruction. `http://www.invisiblethings.org/papers/redpill.html`, 2004.

[30] G. Security. GhostWall FireWall. `http://www.ghostsecurity.com/ghostwall/`.

[31] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *6th ACM/USENIX Symposium on Operating System Design and Implementation(OSDI)*, December 2004.

[32] J. Stewart. Bobax trojan analysis. `http://www.secureworks.com/research/threats/bobax/`, 2004.

[33] C. L. Van Jacobson and S. McCanne. Tcpdump. `http://www.tcpdump.org/`.

[34] VMware. VMware: Virtualization, Virtual Machine and Virtual Server Consolidation. `http://www.vmware.com/`.

[35] C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2), 2007.

[36] Xen. Citrix Xen. `http://www.xensource.com`, 2008.

[37] Xen. Xen v3.0 User's Manual. `http://www.cl.cam.ac.uk/research/srg/netos/xen/readmes/user/user.html`, 2008.

[38] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda. Panorama: Capturing system-wide information flow for malware detection and analysis. In *ACM Conference on Computer and Communication Security (CCS)*, 2007.

Appendix A

Signatures Generated for Bot Families

```
[(’ PRIVMSG #’, ’ :.scanstop -s\n’),
(’:lex!lex@lex.net ’, ’ #lounge ’),
(’lex!lex@lex.net’, ’\n’),
(’ QUIT :Ping timeout’, ’\n’),
(’.anticrew.org’, ’\n’),
(’ PRIVMSG #’, ’ :.asc’, ’ 5 0 ’, ’.x.x’, ’ -s’, ’\n’),
(’.net’, ’ PRIVMSG #lounge :’, ’\n’)]
```

Figure A.1: The signature for one of the behavioral clusters of IRC-1

```
[(’:h3x-’, ’!h3x-’, ’ PRIVMSG ##X## :Accepted’, ’\n’),
(’:Aeolus!evil@feeling.nasty’, ’\n’),
(’ ##’, ’## :.adv.start asn ’, ’0 ’, ’ 0’, ’ -b -s\n’)]
```

Figure A.2: The signature for one of the behavioral clusters of IRC-2

```
[(’ TOPIC #dc :xvvv asn139 150 0 0 -b -r -s’, ’\n’),
(’ QUIT :Connection reset by peer\n’,),
(’:SW!~h4cktsInt@room PRIVMSG #d’, ’ :x’, ’\n’),
(’:SW!~h4cktsInt@’, ’ MODE #d’, ’ +o SW’, ’\n’),
(’.479B98E9.IP MODE #dial# +o ’, ’\n’)]
```

Figure A.3: The signature for one of the behavioral clusters of IRC-3

```
[('@admin.', ' PRIVMSG #', '# :.advscan ', ' -', '\n'),
('!TsInternetUser@', '.com', '\n')]
```

Figure A.4: The signature for one of the behavioral clustera of IRC-4

```
[('CONNECT ', '.', '.', ':25 HTTP/1.0\n'),
('\x05\x01\x00\x03', 'm', '.', 'o', 'o', '.com\x00\x19\n'),
(':eheh!Y@hoo.net PRIVMSG #rs2 :=', '6i', '2', 'W', 'Lb', 'L',
 'o', '4', 't', 'cX', 'k', '5', '\r\n'),
(':', '!Y@hoo.net PRIVMSG #', ' :', 'ps', 'a', 'n', 'o', 'm', '\n'),
(':eheh!Y@hoo.net PRIVMSG #rs2 :=', '6', 'm', 'a', 'F2', 'O', 'x',
 'UO', 'f', 'v', 'S', 'M', 'C', 'm', '\n'),
(':EH!Y@hoo.net PRIVMSG #rs :=', 'f', 'a', 't', 'gA', 'M', 'O',
 'k', 'N', 'A', 'm', '9', 'B\r\n')]
```

Figure A.5: The signature for one of the behavioral clusters of an IRC bot which has an obfuscated C&C.

```
[('HTTP/1.1 .*\r\nDate: .*, 2.* Jan 2008 .* GMT\r\nServer: Apache\r\n.*on.*:
 .*\r\nContent-Length: .*\r\nContent-Type: text/html.*\r\n\r\n<.*HTML.*ET.*
 HT.*EN.*">\r<.*>\r<.*bo.*>\r<.*me.*="h.*om.*">.*</.*>\r</.*>\n.*),
(.*CONNECT smtp.*.google.com:25 HTTP/1.0\n.*),
(.*:tis!yano@admin.siwatech.com .* #siwa.*\n.*)]
```

Figure A.6: The signature for the inbound traffic of the HTTP bot

```
[(.*GET /reg?u=1.*&v=187&s=.*&su=.*&p=0&e=0&o=0&a=0&wr=75 HTTP/1.1\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)\r\nHost:
.*.org\r\nCache-Control: no-cache\n.*)]
```

Figure A.7: The signature for the outbound traffic of the HTTP bot

```
[('P\xfe\x00', '\xe3\x0f\xee1\xd6"\x8a.\x94>\x13\x87n\xd8}\xd4W',
 'Q', "\xeeQ\x19'\xb5o\x96G\xf0\xb5/=)\xf3B\xfc;^H\x0b\x97C"),
('?\xb8\x00', '\xe3\x11\x1aQ\xde@\xa4H\xaeT.\xa1\x9a\xe4\x85\x90Mo',
 '\x01\x00\x00\x00\x02\x01\x00\x01', '.mpg;size='),
('P\xfe\x00', '\xe3\x0fv\xbfj\xb5\x19\xbd#\xc9\xa1\x16\xf6^\x03D\xc2\xe4'),
('P\xfe\x00', '\xe3\x11', '\x01\x00\x00\x00\x02\x01\x00\x01', '.mpg;size='),
('\xe3\x0f\x1aQ\xde@\xa4H\xaeT.\xa1\x9a\xe4\x85\x90Mo',),
('\xe3\x0f\xb7\x00\xab\xf6Z\xfed\r\xe2W7\x9fD\x85\x03\%',)]
```

Figure A.8: The signature for storm