DEVELOPMENT OF A MULTI-LAYERED BOTMASTER BASED ANALYSIS FRAMEWORK

by

Napoleon Cornel Paxton

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Information Technology

Charlotte

2011

Approved by:

____

Dr. Gail-Joon Ahn

____

Dr. Mohamed Shehab

____

Dr. Cem Saydam

____

Dr. Nickolas Stavrakas

ABSTRACT

NAPOLEON CORNEL PAXTON. Development of a multi-layered botmaster based analysis framework. (Under the direction of DR. GAIL-JOON AHN)

Botnets are networks of compromised machines called bots that come together to form the tool of choice for hackers in the exploitation and destruction of computer networks. Most malicious botnets have the ability to be rented out to a broad range of potential customers, with each customer having an attack agenda different from the other. The result is a botnet that is under the control of multiple botmasters, each of which implement their own attacks and transactions at different times in the botnet. In order to fight botnets, details about their structure, users, and their users motives need to be discovered. Since current botnets require the information about the initial bootstrapping of a bot to a botnet, the monitoring of botnets are possible. Botnet monitoring is used to discover the details of a botnet, but current botnet monitoring projects mainly identify the magnitude of the botnet problem and tend to overt some fundamental problems, such as the diversified sources of the attacks. To understand the use of botnets in more detail, the botmasters that command the botnets need to be studied. In this thesis we focus on identifying the threat of botnets based on each individual botmaster. We present a multi-layered analysis framework which identifies the transactions of each botmaster and then we correlate the transactions with the physical evolution of the botnet. With these characteristics we discover what role each botmaster plays in the overall botnet operation. We demonstrate our results in our system: MasterBlaster, which discovers the level of interaction between each botmaster and the botnet. Our system has been evaluated in real network traces. Our results show that investigating the roles of each botmaster in a botnet should be essential and demonstrates its potential benefit for identifying and conducting additional research on analyzing botmaster interactions. We believe our work will pave the way for more fine-grained analysis of botnets which will lead to better protection capabilities and more rapid attribution of cyber crimes committed using botnets.

ACKNOWLEDGEMENTS

During my time at UNC Charlotte, I have been truly blessed to work with an amazing group of people. Without these people this work would not have been possible. My deepest gratitude goes out to my advisor, Dr. Gail-Joon Ahn who has guided me on my Ph.D. journey. His constant inspiration, encouragement, and insightful advice has been invaluable throughout my Ph.D. study. He has not only had an enormous impact on my professional development, but also on my personal life and for that I am very thankful. I would also like to thank Dr. Mohamed Shehab for all his hard work during my Ph. D. study. Dr. Shehab's advice and hands on demonstrations were also a valuable asset that significantly aided my growth during my Ph.D. study. Also, I would like to thank the other members of my committee, Dr. Cem Saydem and Dr. Nickolas Stavrakas for their interest in my work. Their insightful comments and suggestions have truly improved the quality of my work. I have been very fortunate to work with a great group of researchers in the LIISP lab. I would like to thank Moo Nam Ko, Wenjuan Wu, Jing Jin, and Hongxin Hu. Meeting and studying with you all has truly enriched my experience at UNC Charlotte. Thank you and I wish you all the very best in all your future endeavors. I would also like to thank all the students that participated in the UNC Charlotte Cybercorp during my tenure there. I enjoyed the hands on security training we practiced together and am truly appreciative of the hard work many of you dedicated to developing tools that went into the botnet monitoring projects. In particular I would like to thank Jonathan Peterson, Chris Nunnery, Kevin Pearson, David Stone, Connie Kellen, Aaron Friedman, Steven Blanchard, Vikram

Sharma, Jonathan Lavender, and Richard Kelly. These people worked closely with me on various network security problems that significantly enriched my knowledge of network security. I would like to thank my family for their support and encouraging words during good and bad times. I love and appreciate you all. In particular I would like to thank my son Jeremiah. He has always been a major source of my inspiration and always has a positive and uplifting thing to say about his dad. I would like to thank my mom. She has been a true example of strength and she has always believed I was destined for greatness. I only hope to continue to make you proud. I would also like to thank my father, whom I lost during my Ph.D. journey. He showed me what it meant to strive to be the best at what I do and for that I will always be grateful. I would like to thank my grandparents and in particular my grandmothers who both taught me to always trust in God. I would like to give a special thanks to my wife Daelena. Her patience and understanding has been invaluable to me completing my Ph.D. study. Her love and kindness has been a constant motivating factor and I look forward to what life has in store for us. Finally I would like to thank my Lord and Savior Jesus Christ for his grace and mercy which has allowed me to matriculate through my Ph. D. study. Without His love I am nothing and I owe all things to Him.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

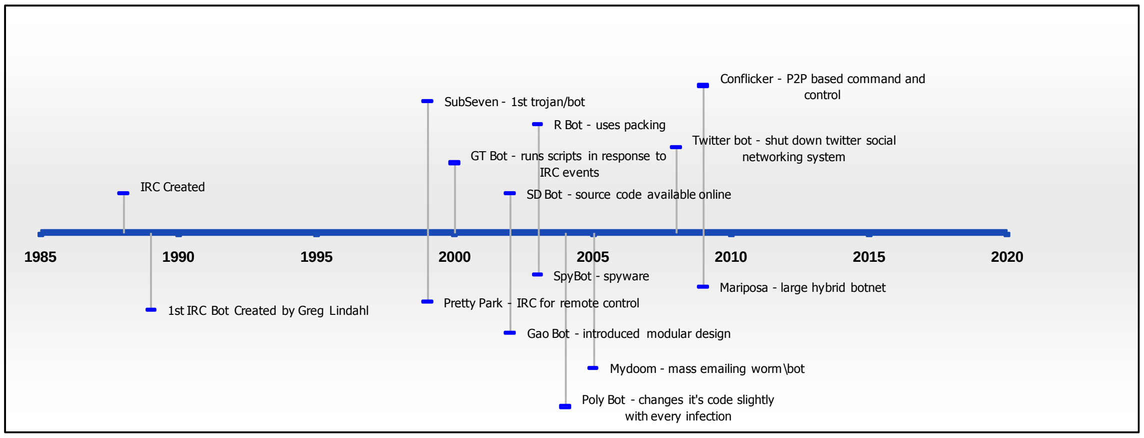
Computer networks and in particular the Internet, have become an integral part of life today. In most cases these networks are enriching and add value to our lives, but because the Internet is connected to most of the world there is also opportunity for cyber criminals to cause significant damage to those that depend on its services. In most cases security services are in place to dampen the threat of cyber criminals, but these services are far behind criminal capabilities in the realm of protecting clients connected to the Internet. One of the greatest threats to the Internet is the botnet. Botnets are networks of compromised machines called bots that carry out the commands of botmasters through communication mediums–such as the Internet Relay Chat (IRC), Peer-to-Peer (P2P), social networks, and so on–which has a command and control that exchange all commands with the botnet. The purpose of a malicious botnet is to facilitate some sort of cyber crime such as spamming attacks, phishing attacks, identity theft, and distributed denial of service attacks (DDoS). Current techniques to study and analyze botnets with the goal of thwarting botnet attacks are insufficient, as is evident by the rising number of botnet attacks and botnet activity throughout the Internet today. These techniques are focused mainly on analyzing the malware which infects the computers and turns them into bots or shutting down the command and control architecture which is the protocol used by the botmasters to command the botnet. While these components of the botnet are important to analyze and understand, we believe there is a glaring omission to botnet analysis that must be addressed. This omission is the analysis of the botmaster himself. In this thesis we address this present deficiency in botnet analysis by sharing our experiences with using our framework to analyze botnets based on the botmasters which control them. For the remainder of this chapter we first discuss the history of botnets and how they became such a threat to computer networks today. We then discuss how botnets and cybercrime in general have had a significant economic impact on society today. We follow this up with a discussion on honeynets, a section on botnet monitoring, a section on botnet analysis, and then we discuss how botnets exhibit social coordination much like other networks. Next we discuss our goals followed by a summary of our research which includes the articulation of our contributions and the structure of the rest of this thesis.

1.1 The History of Botnets

Botnets have a short but destructive history. The bot was originally created for legitimate purposes, but it did not take long for cyber criminals to realize the power of the bot and to discover how they could use it for malicious purposes. Figure 1 is a timeline of the history of the bot beginning from the invention of IRC.

1.1.1 The origin of botnets

Malicious botnets are derived from early programmers creating a way to command agents on IRC chat channels. These programmers would create agents that would do their bidding in a coordinated fashion. The first bot was reportedly invented in



Conflicker - P2P based command and control
SubSeven - 1st trojan/bot
R Bot - uses packing
Twitter bot - shut down twitter social networking system
GT Bot - runs scripts in response to IRC events
IRC Created
SD Bot - source code available online
1985
1990
1995
2000
2005
2010
2015
2020
SpyBot - spyware
Mariposa - large hybrid botnet
1st IRC Bot Created by Greg Lindahl
Pretty Park - IRC for remote control
Gao Bot - introduced modular design
Mydoom - mass emailing worm\bot
Poly Bot - changes it's code slightly with every infection

Figure 1: Bot History Timeline

1989 and was called GM by inventor Greg Lindahl. The purpose of the bot was to play a game of Hunt the Wumpus with other IRC users. Soon after the first bot was created, other programmers realized bots could be used to perform more useful tasks such as administering channels and taking requests from users. During the late 1990s a couple of malicious bots had come on the scene (Pretty Shark, SubSeven). These bots showed future hackers how bots could be used to gain administrative control over infected systems. In 2000, a bot named ”GTbot” was introduced. This bot displayed the ability to run scripts on IRC servers and could conduct flooding attacks which could be used in denial of service attacks. Also in 2000, a sixteen year old Canadian hacker by the name of ”mafiaboy” used a botnet to cause over 1.5 billion in damage to several large e-commerce sites. In 2002 the botnet problem escalated due to the creation of the SD bot. The author of this bot released the source code to the web which made it easy for other hackers to access and customize it for their nefarious purposes. This act of sharing malicious code began the explosion of botnet attacks worldwide.

1.1.2 Newer forms of botnets

IRC based botnets are still prevalent and destructive which is evident by the recent attack on Twitter and other social networking sites [101]. Recently there has been a shift from the more prevalent IRC based botnet to newer and potentially more destructive forms of botnets. These botnets include Peer-2-Peer (P2P) such as conflicker which is described here [68, 43], http, and hybrid botnets such as the Mariposa botnet which consisted of over 13 million computers in 190 countries [19, 82, 33, 5]. P2P botnets use the P2P protocol to command the botnets. In this type of architecture the botmaster can use any compromised peer in the botnet to command the bots. This eliminates the single point of failure problem in IRC based botnets. Http based botnets receive their commands from a number of different URLs over http. In other words the bots in this architecture frequently connect to a pre-programmed list of URLs in order to receive their orders. Hybrid based botnets use a combination of the techniques to command the botnet. An example of this type of botnet could be one that uses P2P to spread and recruit more bots, but connect to certain predefined URLs to download commands. Although these types of botnets are generally thought of as more destructive, a recent resurgence in IRC based botnets shows the need for more research to be done on this type of botnet architecture as well.

1.2 Economic Impact of Botnets

Despite the increased attention to cybercrime over the past few years, financial damage caused by botnets continue to climb. The recent findings of the FBI IC3 2010 report shows that company losses from cybercrime rose from 264.6 million in

2009 to 559.7 million in 2010 [32]. Botnets are the major culprit in these crimes since they are the tool of choice for cyber criminals to use. Botnets have proven to be an effective and flexible tool that the cyber criminals can rent or create on their own. They are also very cost effective since renting or owning a botnet costs far less than trying to defend against them. Some of the crimes committed using botnets are: Distributed Denial of Service attacks (DDoS), identity theft, and spam.

1.2.1 Renting botnets

One way cyber criminals profit from botnets is through renting the botnet out to other criminals that have targets in mind but do not have the technological expertise to design or administer the botnets. This results in a variety of attacks from one botnet by multiple criminals but each with a different intent. Since each botmaster has his own agenda, the transactions between each botmaster and the botnet will vary. Currently these differences in transactions are not normally considered in automatic machine based analysis. This means that most analysis consists of manually combing through logs and analyzing the botnet data as a whole instead of the botmaster being automatically analyzed by a computer program. The result is slow error prone analysis when trying to decipher one botmaster from another which also results in a low rate of convictions when it comes to crimes committed using botnets.

1.2.2 Offense vs. defense

Obtaining and operating a botnet are relatively inexpensive, as opposed to defending against them. It is estimated that those who rent botnets usually pay between $50 to a few thousand dollars for 24 hours of continuous operation of a DDoS enabled botnet depending on the robustness and the functionality of the botnet[58]. Once the cyber criminal has access to the botnet, the recruitment of more bots is free since they infect computers without the knowledge of their owners. Some more technologically savvy cyber criminals can actually download code to create botnets free over the Internet. There are many sites that provide this service and some even provide tutorials which give step by step instructions on how to create and administer the botnets. Because of this ease of access to botnets, the price for defending against the botnet has always been much greater.

1.2.3 Types of attacks

Botnets can conduct a wide variety of attacks against machines connected to the Internet. Here we discuss a few of the more destructive and widely prevalent forms of attacks utilizing botnets.

1.2.3.1 DDoS

DDoS is the most destructive of the attacks committed using a botnet. Servers are created to handle a set amount of connections from customers. When this connection amount is exceeded it leads to a denial of service to some or all of the users trying to access the service. Alone each bot is relatively harmless. It acts as a single user trying to access a service on the server. This is also the case when dealing with a SYN flood attack from one bot. Most servers will be able to handle the attack from a single source because it has the resources to withstand its magnitude. The power of botnets are in their coordination and the volume of the responses from the bot nodes. In a typical botnet hundreds to thousands of bot nodes coordinate to respond to botmaster commands. When these nodes are instructed to connect to one web-service, the aggregated volume of the bandwidth is too much for most companies to handle, causing a denial of service to the targeted servers. Most companies do not have the millions of dollars it takes to properly secure their servers from these types of attacks. Since the denial of service attack comes from multiple bot nodes which are geographically distributed, this type of attack is normally referred to as a distributed denial of service attack. These attacks are prevalent today and cyber criminals are compensated in a variety of ways. Some use the threat of a DDoS attack to extort money from a company [24]. Some perform these attacks to cripple a competitor which will give them a better chance to gain market share and increase their money flow in that way. Others use DDoS attacks as a form of revenge or political statement [60, 3, 64, 81, 34, 13, 38]. These attacks are also growing in nature and analyzed more closely in [26, 45, 13, 14, 62, 63, 1, 22, 94, 61, 59]. In any case, damage from a DDoS attack can be very significant and can cost a substantial amount to repair.

1.2.3.2 Identity theft

In some cases botnets are used to steal confidential information from the true owners of the bot nodes they have compromised. When bot nodes are instructed to download files, the botmaster can receive files from every bot in the botnet which can be hundreds or thousands in some botnets. These downloaded files can vary from secret files that can take away the competitive advantage of the owner, or they can be banking credentials which give the botmasters access to the compromised machine owner bank accounts. In either case, the financial loss for the owner of the compromised machine can be great. This is discussed in more detail in [24].

1.2.3.3 Spam

Another problem caused by botnets that is growing exponentially is e-mail spamming. According to a report by M86 Labs [51], spam typically represents around 80-90% of all inbound Email to organizations which calculates to about 200 billion messages per day which seriously clogs networks, and most of the spam in the world today comes from botnets. Nowadays spam is not only a network-clogging problem, but also a means for criminals to distribute additional malware to infect a greater number of computers when the person reading the spam clicks on the email. This has also become a popular way for botmasters to recruit more bots to become part of their botnet army.

1.3 Honeynets

Although botnets are very complicated to code initially, the creation of botnets has been made relatively easy. The code to program bots have been made modular and are made available to anyone with Internet access. This results in a network full of botmasters which do not necessarily have a strong technical background [87, 30]. Because of this, most botmasters can easily be monitored without them realizing it. A honeynet is a network of machines that are set up to look like a normal network, but have a set of known vulnerabilities. When these vulnerabilities are exploited, these systems are designed to monitor the nefarious activities that take place on them. Using this method we are able to discover the initial compromise of the system and also the commands that are given to the honeynet by the compromiser. In other words this allows us to monitor the interactions between the attacker and the compromised machines. As for the more sophisticated botmaster, more sophisticated honeynets need to be created which can trick the botmaster into thinking he is compromising a real network. This can be very expensive to setup, but depending on the target and the situation it may be a worthwhile endeavor.

1.4 Botnet Monitoring

Since there is normally a geographical difference between the botmaster and most of the bots in a botnet, the botmaster has to command his botnet using a command and control component. This component has evolved over the years, but one thing is still true regarding the connection of the bots to the command and control. Botnets require the information about the initial bootstrapping of a bot to a botnet to be public so this can be recorded and duplicated which allows us to create our own bots which can join the botnet and monitor its transactions. Botnet monitoring has proven to be an effective method to infiltrate and monitor botnets to garner in-depth information about the threat of botnets. Monitoring botnets give us the capability to learn many statistics about botnets, such as the size, the amount of attacks, and how active the botnet is as a whole. Organizations such as Shadowserver do a lot of botnet monitoring to generate these statistics. When the packet capture traces are recorded between the bot and the botmaster there is a unique opportunity to view the messages sent between the botmaster and the bot. Since botnets are normally massive in size, it has been relatively easy to covertly infiltrate a botnet and monitor its transactions. Because of this, botnet monitoring has become a common way to analyze and identify botnets and the destruction they cause. The idea behind botnet monitoring is to capture a bot, modify the bot so that it will not become part of any subsequent attacks, allow the bot to connect to its command and control center, and then monitor the communications that take place on the botnet. This works well when trying to explain the magnitude of the botnet problem as a whole, but a finer grained analysis technique is still needed to develop better protection mechanisms to defend against discovered threats. In this research we extend botnet monitoring techniques based on the interactions between botmasters and their botnets. Figure 2 shows the basic operation of an IRC botnet. This figure addresses a deficiency present in previous figures which only show a botmaster sending commands to a botnet, giving the impression that there is only one botmaster in control of the botnet. In fact, most botnets are controlled by multiple botmasters. In the figure we see botmaster 1 initially creating the botnet. Once the botnet is created, botmaster 1, 2, and N (which represent all other botmasters commanding the botnet) each have a different attack agenda. Discovering these agendas and the roles played by each botmaster is the goal of this research.

1.5 Botnet Analysis: New Focus

Once we have the monitored data as an input, most analysis research in the area of botnets are focused on finding bots and command and control channels and shutting the botnet down quickly [2, 27], but as Park and Reeves pointed out [68], it is also important to monitor botnets for an extended time to learn the purpose of the botnets to develop more effective countermeasures. A large problem with slowing the rise of

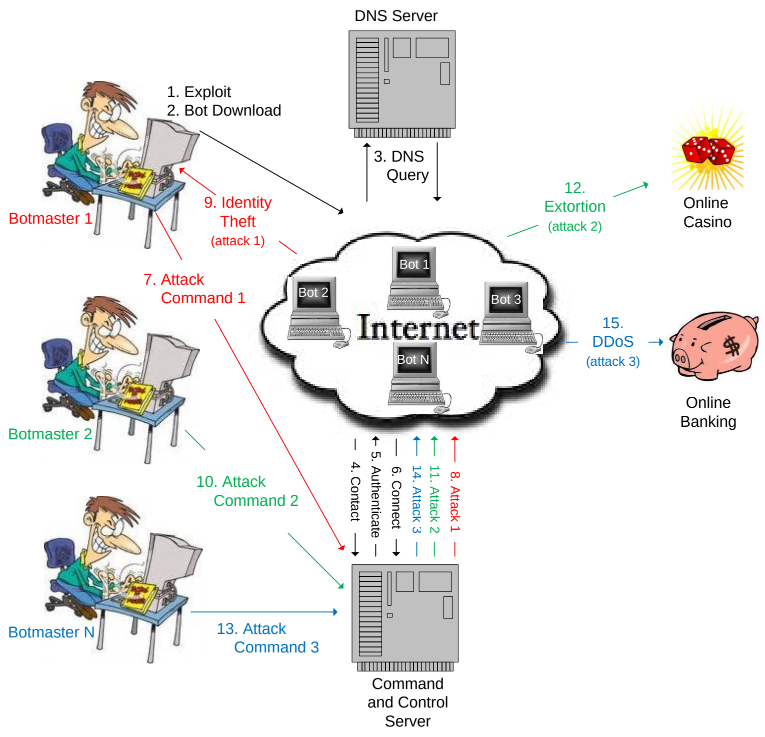


Figure 2: Basic Botnet Operation

botnets is that most significant botnets today are constantly changing. They are evolved by adding bots, deleting bots, changing to new channels, being upgraded, etc. In order to stop botnets we need a sophisticated analysis method, instead of relying on only attempting to discover their command and control servers. This can only be achieved by conducting a long term analysis [59]. A good example of this is what happened after the take down of the largest spamming botnet in the world, the McColo botnet. In November 2008, the most major spamming botnet in the world, McColo was shut down. The next day the spam volume was nearly cut in half, but by the end of 2009 the volume was increased higher than ever. Experts agree that the explosion in spamming is a result of the botmasters in charge of McColo regrouping and creating other botnets in which they could spread their spam once again [51]. In order to address this issue we focus on discovering characteristics of the botmaster over time which will allow us to not only discover the means to shut the botnet down, but hopefully enough information to discover the attacker to prevent the creation of potential botnets.

1.6 Social Coordination

We argue that botnets can be looked at as a social network. Social networks tie nodes together based on the interdependency. The way we look at these interdependencies are as commands and their semantics. We see this often in networks of people. Suppose we have a network (group) of house builders. In order to build a house we have a foreman that gives the commands (orders) to builders. At times the foreman tells a subset of the builders to lift a wall into place, and at other times he tells all the builders to go to the next house or go take a break. There are even situations when the foreman will tell one worker to install a light bulb or order one of the workers to leave. In botnets the botmaster plays the role of the foreman and the bots play the role of the workers. The botmaster can command the bots to download software updates, or attack a victim website. He could order one bot to leave the botnet, or a subset of the bots to download banking credentials from their compromised hosts. In our work we begin to explore modeling the social aspect of the botnet by analyzing the transactions between the botmasters and the bots on the botnet.

1.7 Botmaster Based Analysis: Our Goals

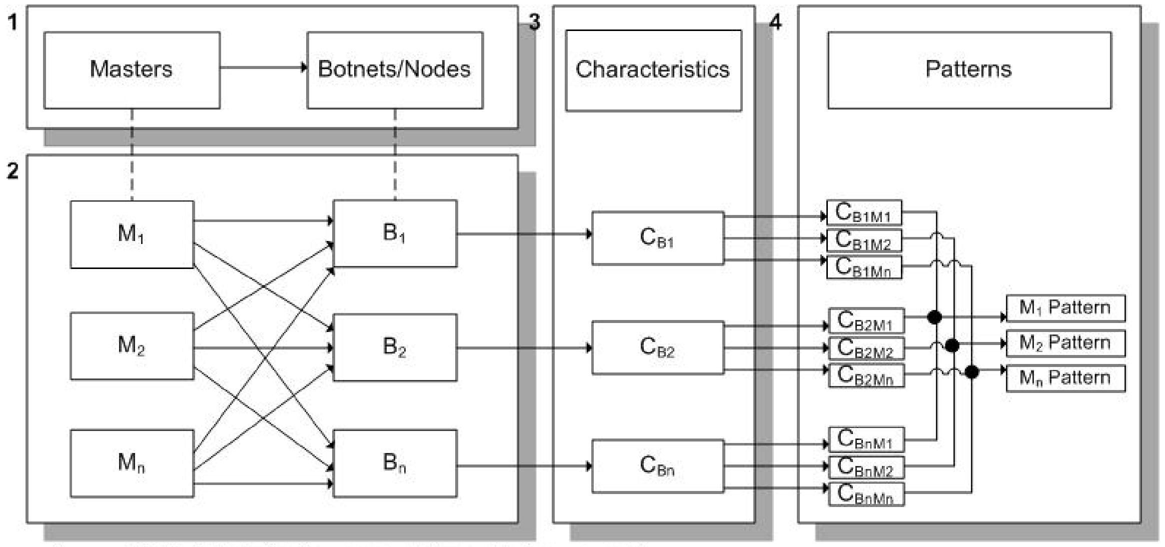
Current analysis techniques have done a good job of enumerating the size of botnets and how they are organized. These techniques concentrate on the botnet as a whole and therefore omits the transactions between the botmaster and the bots. They also do a great job of analysis on individual bots. In our earlier research we also performed bot based analysis as shown in [4, 70, 71]. We also describe these techniques in chapters 2, 3, and 4. In order to eventually stop botnets we have to hold those that are at fault accountable. In order to do that we need more fine grained analysis of the botnet transactions. In particular we need to concentrate on botmaster based analysis. This type of analysis presents several issues:

1. **Storage of packet captures can get very costly.** Since the transactions between the botmaster and the bots within the botnet are located in the payload section of the network packets, we capture the full packet capture traces (PCAP) of the botnets. These PCAPs consume a lot of storage space and requires a large amount of computational power to analyze. In order to alleviate some of this burden we parse the PCAPs for the usable content and then discard the rest. In the future we plan on modifying current network flows to also capture the application layer payload data since network flow data takes much less storage and is more efficient to analyze.
2. **All transactions are not displayed in plain text.** In this case we have two choices. We can either try to decipher the transactions using various decrypting techniques or we can omit the cryptic results. It is our aim that by showing the benefit of analyzing the botmaster transactions we will inspire more research in decrypting more advanced botnet transactions.
3. **Botmasters do not generally use their real name or other personal attributes when connecting to a botnet.** Since botmasters do not want others to know who they actually are, they normally do not include any identifiable information in their transactions with botnets. Some botmasters even use multiple botmaster names when they are commanding their botnets just in case someone is starting to suspect who they might be. This is not a real problem in our work. We aim to identify what the botmaster names are doing, so we do not concentrate on the actual person behind the transactions at the moment. It is however possible to correlate multiple botmaster names to determine if they are exhibiting similar behavior. We plan on working on that issue in later work.

In our research we have the following goals:

1. **Discover botmaster characteristics.** Using monitored botnet data as input, identify transactions between each botmaster and the botnet. We aim to discover all the exchanges of information within the botnet and attribute it to the botmaster that is initiating the transactions.

2. **Discover the level of impact of a botmaster within the botnet.** Here we aim to discover the threat each botmaster poses based on his involvement in the botnet. We also aim to see what types of activities the individual botmaster participates in the most.



1. Initial install of reflective commands (parsed bot source code)
2. Botmasters sending impulsive commands to botnets and the nodes within the botnets (PRIVMSG transactions from Masters (M1) to nodes in a Botnet (B1)
3. The impulsive commands matched with the reflective commands in each botnet equal the associated links which are also the characteristics (reflective commands found in PRIVMSG transactions)
4. The aggregate characteristics for each botmaster is the botmaster pattern (All Characteristics (C) found in every botnet (B) for each Master (M)

Figure 3: Research Solution

Figure 3 is a diagram that shows the methodology of how we turn botnet transactions into categories that are botmaster based. In step 1 of the diagram we show that one botmaster initiates the botnet process by creating the first botnet nodes. The binaries of these nodes are analyzed in our analysis system which is described in the next chapter. Once the nodes are analyzed, they are parsed and reflective keywords are extracted. These keywords are the first building blocks to characteristics which are described in detail in chapter 5. In step 2 one or more botmasters send commands to the nodes of the botnets. We call these commands impulsive commands and they are also described in chapter 5. In step 3 these reflective keywords and impulsive

Table 1: Framework Component Details

Component	**Requirements Met**	**Description**
Bot Capture	Characteristic Discovery	Starts the analysis
Closed Analysis	Characteristic Discovery	Scans bots
	Characteristic Discovery	Discovers bot keywords
	Characteristic Correlation	Submitted for correlation
Open Analysis	Characteristic Discovery	Connects to node hosts
	Characteristic Correlation	Matches correlated
Net Monitoring	Characteristic Discovery	Captures transactions
	Characteristic Correlation	Matches correlated
Correlation	Characteristic Correlation	Correlate results of components
	Characteristic Correlation	Discover social characteristic type
	Characteristic Correlation	Discover evolutionary characteristics
	Characteristic Correlation	Match botnet map characteristics
	Characteristic Correlation	Generate correlation
Threat	Threat Generation	Discover significance level
Taxonomy	Threat Generation	Generate significance level
	Threat Generation	Generate taxonomy
Protection	Threat Generation	Generate report

commands are combined to create characteristics and in step 4 patterns based on the botnet are created from these characteristics.

1.8 Research Overview

To accomplish our goals we have developed a framework composed of eight components. Alone each component provides its own results which are a low level of analysis given the correctly structured input. Collectively each component combines with each other to provide multiple layers of analysis which give a detailed output. In our framework each of the eight components satisfy at least one of the three system requirements. Figure 4 displays our framework. Table 1 shows the details of each component.

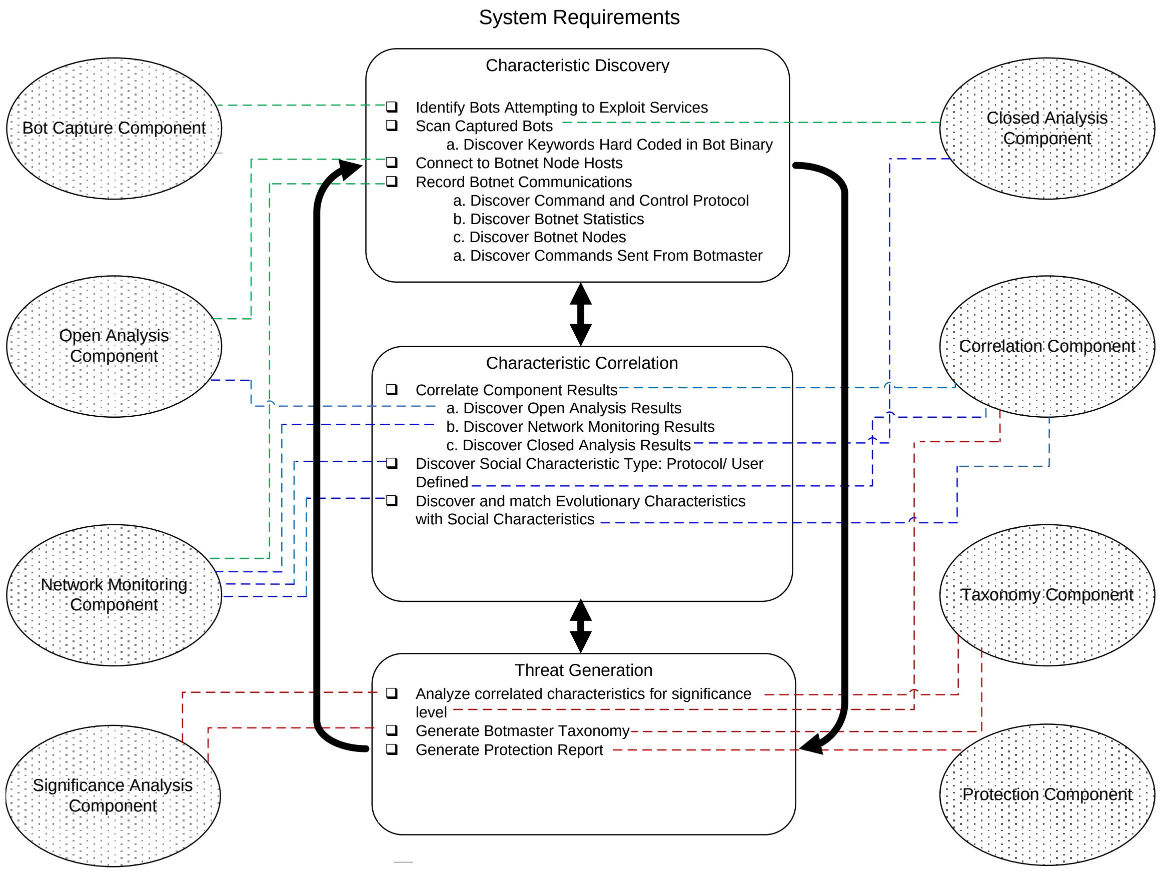

System Requirements
Characteristic Discovery
Identify Bots Attempting to Exploit Services
Scan Captured Bots
a. Discover Keywords Hard Coded in Bot Binary
Connect to Botnet Node Hosts
Record Botnet Communications
a. Discover Command and Control Protocol
b. Discover Botnet Statistics
c. Discover Botnet Nodes
a. Discover Commands Sent From Botmaster
Characteristic Correlation
Correlate Component Results
a. Discover Open Analysis Results
b. Discover Network Monitoring Results
c. Discover Closed Analysis Results
Discover Social Characteristic Type: Protocol/ User Defined
Discover and match Evolutionary Characteristics with Social Characteristics
Threat Generation
Analyze correlated characteristics for significance level
Generate Botmaster Taxonomy
Generate Protection Report
Bot Capture Component
Open Analysis Component
Network Monitoring Component
Significance Analysis Component
Closed Analysis Component
Correlation Component
Taxonomy Component
Protection Component

Figure 4: Research Framework

1.8.1 Characteristic discovery

Characteristic discovery is concerned with discovering the building blocks that will be correlated to become characteristics. Within the characteristic discovery requirement we have four main tasks. Our first task is to **identify a bot attempting to exploit services**. This is done using the bot capture component. The goal of this task is to covertly copy a bot of interest and not be detected by the administrators of the botnet. The next task is **scan captured bots**. This task is completed within the closed analysis component. The goal of this task is to discover the commands and keywords hard coded in the bot binary. The task **connect to botnet node hosts** is performed by the open analysis component and its goal is to create a connection between the captured bot and the botnet. The task **Capture Botnet Communications** is carried out by the network monitoring component. In this component the network traffic captured after setting up the communication line in the previous task is analyzed for characteristic building blocks. All these elements become the foundation of our analysis system.

1.8.2 Characteristic correlation

Now that we have discovered the individual elements of each characteristic from the botnet traffic, we need to correlate them into characteristics to make the information meaningful. The first task is to **Correlate Component Results.** The components involved with this task are closed analysis component, the open analysis component, the network monitoring component, and the correlation component that performs the matching. The purpose is to discover evolutionary change at the botnet level and also discover the social characteristics at the botmaster level which exists within the channels of the botnet. The next task is to **Discover Social Characteristics Type: Protocol/User Defined** based on protocols defined from RFCs and user defined criteria. Using the social results, the purpose for this task is to discover how involved a human is in the command generation process. A bot node that generates more User Defined characteristics within his communications is generally responding to human initiated commands verses the more commonly found generic bots that conduct protocol based commands more often. Next we **Discover and Match Evolutionary Characteristics with Social Characteristics**. Here we discover how the size of the botnet corresponds with the social activity of the botnet and discover whether or not it has an effect on botnet attacks.

1.8.3 Threat generation

To discover the threat involved with the botnet transactions, we take the correlated results as the input. The first task to be completed is to **Analyze Correlated Attack Matches for Significance Level**. This gives us an idea of what role each botmaster plays in the botnet. The significance analysis component, correlation component, and taxonomy component handles this task. The next task to be completed is to **Generate Taxonomy of Botmasters**. Here the generator or "master" of the botnet traffic is categorized. Each time a new characteristic is discovered to be generated from a master, the masters taxonomy entry is appended. This task is carried out by the significance analysis component, taxonomy component, and the correlation component. The next task is to **Generate Protection Report**. In this task when an administrator decides to do an analysis of the botnet, he can generate a protection report. This report alerts the administrator about the threat each botmaster is posing to the network. This report is carried out by the protection component.

1.9 Research Contribution and Thesis Organization

In our research, our contributions are manifold as follows: first, within the monitored data we attribute each transaction to the botmaster and categorize the transactions based on a modified version of the reflective-impulsive model [91]. Our reasoning is, although a botnet is destructive, it is still just a tool, and a tool is only as good as the person who uses it. For example, an experienced tailor can use a sewing machine to sew a shirt better than an inexperienced tailor. A tool is also only as good as the way it is used. If a Swiss army knife is always used as a knife and a screw driver then the unused scissors that come with the knife does not matter. Botnets are similar to the sewing machine and the Swiss army knife. Its usefulness is determined by the skill level of the botmaster and how they correspond with the botnet. We categorize the botmaster interactions as social characteristics since there is a 2-way correspondence between the botmaster and the node in a botnet that responds to him. Second, we identify the evolution of the physical characteristics (size) of a botnet. In most situations the size of a botnet determines the magnitude of the botnet's attack power.[1] When it comes to size, botnets also have the same characteristics as other networks, like human social networks which are constantly in a state of flux. These communication networks are born, grow, shrink, and also disappear. Because of this, we explore botnet evolution to track and garner physical characteristics from each evolutionary stage of a botnet. Third, we correlate the discovered social characteristics and the evolutionary characteristics to shed light on the role and effect each botmaster plays within a botnet. To the best of our knowledge, this is the first attempt to identify the evolutionary characteristics of a botnet, and also to analyze a botnet based on its botmasters. The remainder of the thesis is structured as follows: Chapter 2 discusses the related work and why our approach is needed to provide a higher level of granularity in botnet analysis, Chapter 3 presents our reasons and motivation for developing a honeynet testbed setup and how it provided the groundwork for our research as well as served as an educational tool to explore network security in a testbed environment.

[1]The compromised machine bandwidth is also a major factor of attack power.

Chapter 4 presents our techniques for monitoring botnet attacks. Here we introduce the theoretical development of our framework and show how each component of the framework is needed to accomplish our goals. Chapter 5 presents an introductory view of how we could realize the bot detection and bot monitoring portion of our framework through IRC Sandman which automatically downloaded and studied bots from IRC channels. Implementation and results of IRC Sandman on a botnet are shown here. Chapter 5 also goes into more detail about the remaining components of our framework. Chapter 6 discusses our motivation and experiences developing our botmaster based system which identifies the role of each botmaster within a botnet. Here we present "MasterBlaster" which is the realization of our framework. Implementation and results of the analysis of "MasterBlaster" on three monitored botnets are shown here. Chapter 7 summarizes the lessons learned from the development of our framework and in particular "MasterBlaster", the botmaster based analysis system. This includes a discussion about the limitations and future aim for our research. Chapter 8 concludes this thesis.

CHAPTER 2: RELATED WORK

Our motivation for creating our honeynet based architecture is to create a robust botnet monitoring and analysis system. Botnet monitoring and analysis has remained a hot topic in the last few years due to the continued increase in destruction caused by botnet attacks. Current methods to monitor and analyze botnets have provided valuable information that allows us to discover many interesting characteristics of botnets. The problem is most of the information provided today does not lead to actionable intelligence that can reduce the impact of botnets. Our goal is to extend the current methods by identifying a fine-grain level of characteristics that will lead to developing actionable intelligence. Below are works that relate to our research.

2.1 Honeynet-based Data Capture

Honeynets have proven to be a valuable tool in our research. Many other projects have used honeynets to capture data from unauthorized users. Li, Goyal, and Chen used honeynets to discover capture data [48]. Yegneswaran, Barford, and Paxson used honeynets to model Internet situational awareness [102]. Provos explored deploying virtual honeynets in [77]. In this study he was able to quickly deploy honeynets with little cost or setup. Cai explored using honeynets to defend networks using a game theoretic approach in [11]. All of these projects use honeynets in the same way we do which is to capture unauthorized traffic from an attacker. Our approach differs from theirs in the purpose of the data capture which is to discover botmaster characteristics.

2.2 Botnet Monitoring

Li, Goyal, and Chen used botnet monitoring to discover botnet traffic as we did. The difference is their approach was to analyze scanning traffic, where our approach was conversation centric [48]. Dagon, Gu, Zou, Grizzard, Dwivedi, Lee, and R. Lipton introduced a taxonomy of botnets to provide a response to botnets by degrading or disrupting them [21]. This method involved discovery and proactive attack to the botnet. In our work, we focus on a bot taxonomy composed of properties that generate patterns of the botmasters. Some earlier works addressed issues on tracking botnets [25]. Such works adopted sensors and honeypots to investigate a pathway to and from botnets. Our approach uses a virtual space such as honeypots to capture bots and track botnets. Other projects that utilized botnet monitoring were [78, 79, 49, 105].

2.3 Botnet Analysis

Defending networks against botnet attacks is an ever growing issue in network security and cyber crime research communities. To our knowledge, there are only a few works using threats as a deciding factor such as the McAfee Advanced Botnet Protection in Intrusion Prevention System [23]. This tool uses a proxy to accept or block traffic that appears to be botnet related. It does not use the threat value rigorously but mainly relies on a signature based approach. Our architecture is very similar to the approach as noted by Rajab, Zarfoss, Monrose, and Terzis [2]. Some key differences are that instead of creating (drones) to connect to a command and control, we (install) the actual captured bot on a honeypot to connect to its command and control. Our correlation system component is also a major difference in that we are keeping track of the characteristics of each botmaster. Other research which focuses on botnet analysis includes work by Park, Pai, Lee, and Calo [67], work by Binkley and Singh [9], [8], and others [104, 7, 50, 92, 84]. These projects focused on discovering bots within the network traffic. In our approach we own the bot and are analyzing the traffic between it and the botmaster. Several projects have studied the behavior of bots within the botnet such as [90, 52, 16, 15, 10, 36]. These projects are similar to our approach since they are studying activities of the bots in the botnet. Our approach differs from these in that we analyze the conversations and not statistical characteristics of the bots such as time or spacial deviation which are found in these approaches.

2.4 Attribution

Attribution in our case consists of discovering what attacks belong to what botmaster names. The Internet consists of proxies, and allows for IP and MAC masking among other anonymization techniques which make it very difficult to determine the true perpetrator behind an attack. Issues in obtaining attribution are discussed in more detail in [88, 31]. In our research we attribute the botmaster usernames to the command activity they carry out on the botnets. Research which attempts to perform attribution is normally geared around watermarking of packets such as [69, 97, 72]. These projects differ from our approach by not taking into account the content of the packets which we believe is needed for true attribution.

CHAPTER 3: HONEYNET-BASED TESTBED ENVIRONMENT

Now that we have discussed the problem, articulated our goals for our research, and shown how current research does not fully address the problem by discussing related works, the next order of business is to setup a test bed to capture botnet related network traffic. In this chapter we discuss our experience developing our honeynet-based testbed and how it set the groundwork for the rest of our research. In this chapter, we describe our Honeynet-based Bot Analysis Architecture which includes collecting bots in our **Malware Collection System Component**, running them on an off-line simulated network in our **Closed Analysis System Component**, and installing them on our **Open Analysis System** to connect with their command and control center. We use the actual collected bots to connect to their command and control centers instead of simulated attack bots, sometimes called (drones). We use our analysis template which is our **Correlated System Component**, to discover characteristics of each individual bot. This template has proved to be an invaluable learning tool for students to interact with malware in the wild. The rest of the chapter is organized as follows. Section 3.1 discusses background information. The Honeynet-based Bot Analysis Architecture is presented in Section 3.2. In Section 3.3, we discuss our analysis method along with the results. Section 3.4 concludes this chapter.

3.1 Background

In this section we discuss a brief background of honeynets and why we are utilizing their framework. We also discuss command and control architectures and issues with the current methods to detect botnet attacks.

3.1.1 Honeynets

A honeynet is a network composed of two or more machines that has the sole purpose of allowing itself to become compromised by a cyber criminal. Once the machine is compromised, another machine records the interactions observed between the attacker and the compromised machine all the while this second machine thwarts any attacks the attacker attempts from the compromised machine. This type of architecture has proven very beneficial in learning about cyber criminals and their methods of attack. Honeynets have been used to learn as much about bots and the attacker sending bots as possible [75]. We use this approach, because it provides us with the data between the attacker and the compromised machine. Even though this approach allows us to gather attacker footprints, a systematic data analysis method is still needed.

3.1.2 Command and control architectures

As mentioned earlier, in the botnet the command and control is where the attacker sends commands to the botnet. Currently most malicious bots use IRC to communicate with the command and control. IRC built in multi-cast capabilities make it easy for the commander to send orders to all the bots in the botnet without much effort [47]. We also briefly mentioned earlier that a more destructive form of communication for bots is with the P2P protocol. These bots contain P2P clients and can communicate with one another without the use of a central command center. With this type of command and control the attacker can initiate commands by posing as a peer anywhere in the network. Other forms of command and control are also being used to a lesser degree, such as instant messaging and cellular phones. As researchers continue to find ways to protect against IRC based command and control structures, the number of botnets controlled by other protocols will continue to increase.

3.1.3 Detecting botnet attacks

DDoS attacks are extremely difficult to detect. Most existing mechanisms have limitations to properly distinguish botnet traffic from legitimate traffic, generating a high false positive rate [23]. A high false positive rate may be its own denial of service, since legitimate traffic is blocked from accessing the network. Some research which discuss this problem are [37, 42, 55, 103, 85]. These methods were able to detect botnet traffic at varying degrees, but still suffered from false positives. Botnets will continue to be a growing threat until a trustworthy mechanism is presented that effectively detects and blocks botnet attacks while allowing a very low false positive rate [76, 28].

3.2 Honeynet-based Bot Analysis Architecture

Our honeynet testbed was created to satisfy three major requirements:

1. **Systematically collect and analyze malware traffic over the Internet**
2. **Comprehensively discover characteristics and unique behaviors of malware**

3. **Dynamically determine associated threats and generate corresponding threat reports.**

3.2.1 Architecture components

In this section we discuss the components of the architecture without having any specific tools in mind. Any tool that can perform the tasks described here can be used as part of the architecture. This requirement is to ensure the extensibility of our architecture. Figure 5 is a visual representation of our architecture.

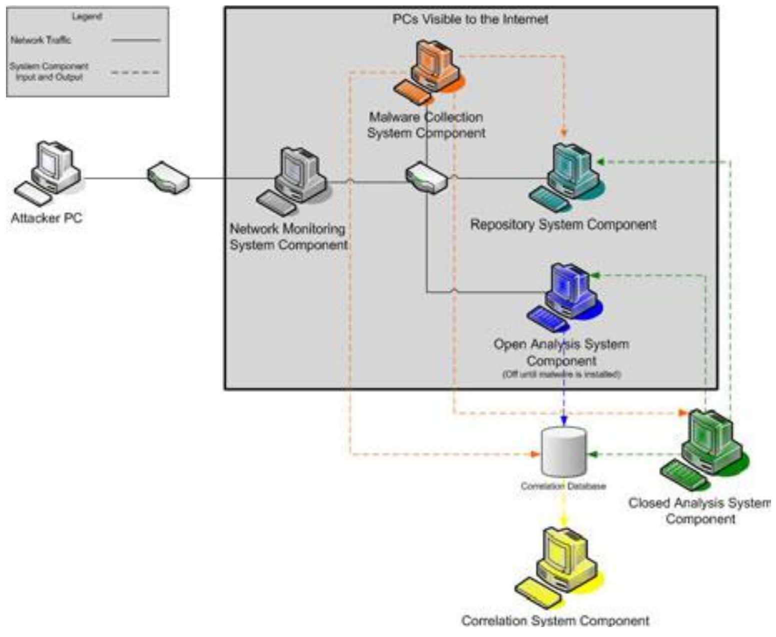

Legend
Network Traffic
System Component Input and Output
PCs Visible to the Internet
Malware Collection System Component
Attacker PC
Network Monitoring System Component
Repository System Component
Open Analysis System Component
(Off until malware is installed)
Correlation Database
Closed Analysis System Component
Correlation System Component

Figure 5: Architecture Diagram

3.2.1.1 Malware collection and network monitoring

Before we can discover what the risks are in a network, we need to discover how attack code reacts with the system. To realize this goal, a collection system is proposed that collects bots to be dynamically analyzed. Dynamic analysis occurs by ensuring that the collection system emulates each of the services on the network it is protecting. We capture bots by emulating the vulnerable services. Also, this system provides protection against significant involvement in attacks after the bot has been run on the system. It uses firewall and intrusion protection techniques, such as limiting or dropping packets leaving the protected network.

3.2.1.2 Closed and open analysis system components

These components take the binary captured in the collection system and runs it on a closed network environment. This is a necessary step to discover certain aspects of the malware before putting it on the open analysis system and opening it up to the network. The closed analysis component has the capability to use attack commands found in the binary and perform simulated attacks using a Perl script. These attacks are only run in the simulated network and will give insight to what the binary is made to be used for. It includes the discovered hard-coded DNS addresses, attack commands, and other functionality of the bot. Eventually more functionality will be identified from the closed analysis such as patterns from the virtually simulated attacks that can be performed within the closed analysis system. The open analysis component of this system allows us to inject a malicious bot into a computer and connect back to its original destination. This enables us to isolate the bot from the network and monitor its traffic in a more controlled way instead of waiting to be infected and then monitoring the traffic passively. The strings are pulled from the binary as it is being run in memory, thereby negating any obfuscation techniques.

3.2.1.3 Correlation and repository system component

The pattern correlation system takes input from the open analysis and closed analysis systems and creates an intelligence report to display the alert events that are identified from the bot installed. This intelligence report is used to discover patterns in the traffic and correlations between logs. The goal of the correlation system is to gather as much information (characteristics) about each individual bot as possible and correlate the results with other bots to discover a record of each bot which is sent to our repository system component. Each bot record will have a list of its own characteristics as well as references to other bots that use or have any connection with the bot entry in the repository. The purpose of the bot record is to provide a comprehensive identity for the bot so the characteristics provided by the identity will lead to an accurate assessment of the risks or threats they present. A record updater is also needed to keep the repository up to date and accurate. When a new correlation is found in a bot, its repository entry will change to reflect the new correlation. All other bot records that are cross referenced by that bot will then be updated by the record updater. The repository is a central collection of all the logs in our architecture. This gives the administrator a macro view of the protection system and provides an aggregated view of the attackers on the network.

3.2.2 Tools used in analysis

In this section we describe how we have analyzed the bots and what tools were used in the components of our architecture.

1. **Network Monitoring:** Used to capture all the transactions between the attacker and the drone machine after we have captured a bot, modified it and

stored it on the Open Analysis System Component. We use one tool to perform this analysis.

- Honeywall: A firewall that allows the connection of an attacker to our analysis system. [76]

2. **Malware Collection:** Component for capturing and storing binaries. We used two tools to accomplish this.
   - Nepenthes. A low interaction honeypot for capturing malware [6, 56].
   - MySQL. Our database of choice for storing the malware [57].
3. **Closed Analysis:** Component for analyzing each captured binary off-line before allowing it to be run in its native environment. To implement this component we utilize one tool.
   - Sandnet. Sandnet emulates the Internet and gives us the ability to act as the command and control by sending commands found in the strings to a python script that allows us to issue the bot commands [95].
4. **Open Analysis:** Component for analyzing each binary in its native environment. We currently use seven tools to perform this analysis.
   - VMWare. This tool gives us the ability to run our bots on an operating system image that can be quickly restored to the previous system state. This allows us to quickly switch from bots to bots in our analysis process [96].
   - Perileyez. A malware analysis tool that compares snapshots of the system and produces all the changes made. We run this tool before we place the bot on the honeypot and to observe any immediate changes it makes [73].
   - Sebek. A root kit used to collect all the system calls from a client and server. We use this root kit to record all the commands given from the bot master to the bot [74].
   - Wireshark. This tool analyzes network packets. We use this as a learning tool to manually analyze packets [100].
   - Honeywall. It monitors all packets in and out of our architecture. It also provides us with data control, which is our fail-safe shutdown method to avoid being an active participant in a botnet attack [75].
   - Maxmind Database. Tool for displaying the location of an IP on a world map. We use this tool to map the source locations of where the malware was downloaded from [53].
   - Norton AntiVirus and ClamAV. We use this tool to determine whether the antivirus signature and categorization for each bot exists [93, 17].
5. **Correlation System Component:** Component for combining data from the other components into an analysis report. We use two tools to perform this analysis.
   - XML based botzoo report. This report is XML based so it can easily be imported into other tools in the future.
   - MySQL. Our database of choice for storing the analysis results from the other components.

3.2.3 Malware interaction

This section discusses the steps of how the malware interacts with our components, including each tool role in our architecture. Malware collection is achieved using Nepenthes, which is a program that emulates Microsoft Windows services to incite automated attacks. When an attack occurs, Nepenthes logs the malicious activities and attempts to download any binaries associated with the attack. The downloaded malware is automatically stored in a MySQL database on the architecture, as well as the originating IP address, and run through two anti-virus engines, Norton 10 Corporate and ClamAV. The anti-virus engine results are then stored in the database. Our Maxmind Database detects the source of the bot and adds an entry on a map of the world to geographically visualize the location of the bot. For our Closed Analysis we use a simulated environment. Although, Norman Sandbox is the most popular malware simulation environment, we use a tool called Sandnet. Sandnet provides an isolated environment and a virtual network for the piece of malware to execute. The environment consists of two computers, a Sandnet Server and Sandnet Client. After the initial execution of malware, an MD5sum file, memory dump file and network traffic logs are sent to the Sandnet Server from the Sandnet Client. Using a specifically designed Perl script we can recompile the memory dump file and run the Linux command (strings /minus a ¡file¿) to obtain the strings off of the malware. The strings allow us to determine commands used by the malware as well as target areas that the malware will be likely to hit. Furthermore, Sandnet is able to simulate various types of servers, the most important being an IRC server since this is the most notorious avenue for sending malware commands. Our Open Analysis provides connection to the Internet. The live execution environment is notably more verbose than using Sandnet. To begin, VMWare workstation is used to create a default installation of Windows XP, Service Pack 1. After the image is created, Sebek is installed onto the image. Sebek is a kernel based data capturing tool and captures the processes used by the image, sending them as packets across the network. To obtain the files added, deleted and changed by the malware the tool Perileyez is run on the image. The initial snapshot of the image is taken once Sebek and Perileyez are installed and after the malware is executed, a second snapshot is taken. By comparing the two snapshots we can identify alterations the malware makes to the image including changes to drivers, DLLs, processes, ports and remote connections as well as any files changed. Capturing and analyzing network traffic is the final step in running a live execution environment. To capture all network traffic generated by the virtual environment, we use a Honeywall. The Honeywall is able to capture all network packets that are sent and received by the image. These packets are merged into PCAP files and sent to a central server at the end of each day. Currently we have found it useful to separate the PCAP files into four hour segments, giving us six slices for each day. By segmenting the file, it allows us to locate suspicious data more easily. Using the tool Wireshark, we can look at the daily PCAP files and determine the actions of the malware for the previous day. One PCAP file can display IRC conversations, secondary injections attempts, DNS queries, propagation scans and HTTP conversations as well as any other type of network traffic.

3.3 Analysis Methods and Results

Here we present the methods we used to analyze the malware we collected with Nepenthes. After the brief discussion about the methods, we present our preliminary results.

3.3.1 Analysis methods

We analyze our collected malware using a predefined method. The list below shows the content of each analysis. Each week information security students share their findings on bot characteristics using this template with other students and faculty. This has greatly increased their competency in analyzing these bots within their native environment.

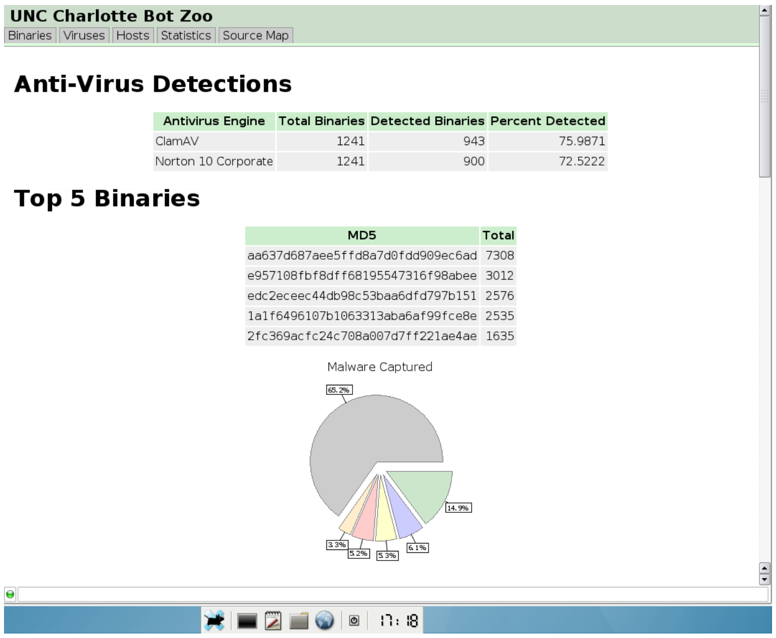
1. **Identification:** MD5 value and anti-virus engine results
2. **Source of Infection:** network traffic analysis related to the location where the malware downloaded from.
3. **System Interaction:** system state report which includes files added, unloaded drivers, unloaded DLLs and so on.
4. **DNS queries:** identification of domain names for command and control servers and corresponding ISP information
5. **IRC Communications:** collection of live IRC conversation and any traffic related to scanning and secondary injection

3.3.2 Results

Here we describe our results generated from our architecture. We first describe some of the statistics of the bots that were captured in the bot capture component and then we discuss some interesting discoveries concerning the analysis made possible by the network monitoring component, the closed analysis system, and the open analysis system. We then show results from the correlation of the bots.

3.3.2.1 Correlation statistics

Most of the malware that we have examined have exhibited similar behavior. Figure 6 is a snapshot from our bot repository. It shows our number of total binaries as opposed to the number of binaries that were actually detected.



Anti-Virus Detections

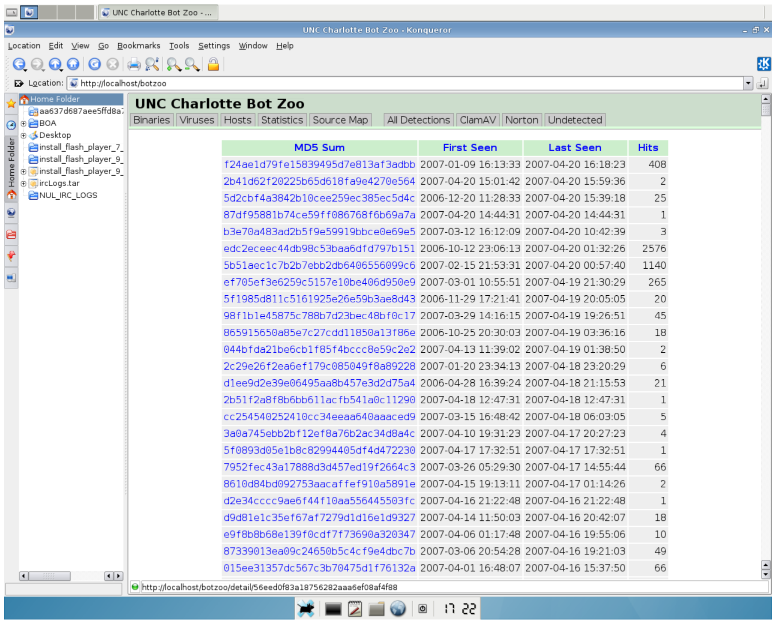
Antivirus Engine	Total Binaries	Detected Binaries	Percent Detected
ClamAV	1241	943	75.9871
Norton 10 Corporate	1241	900	72.5222

Top 5 Binaries

MD5	Total
aa637d687aee5ffd8a7d0fdd909ec6ad	7308
e957108fbf8dff68195547316f98abee	3012
edc2eceec44db98c53baa6dfd797b151	2576
1a1f6496107b1063313aba6af99fce8e	2535
2fc369acfc24c708a007d7ff221ae4ae	1635

Figure 6: Anti-Virus Detection

As we notice, both Norton Antivirus and ClamAV did not detect nearly 25% of the bots that were downloaded in Nepenthes. When started, at least one and as high as fourteen executable were installed on the image. Figure 7 shows the binaries downloaded from nepenthes. As shown some binaries were downloaded over a thousand times. This shows that the attackers are sending out the bots using a script and they do not care about duplicate infections.



UNC Charlotte Bot Zoo

MD5 Sum	First Seen	Last Seen	Hits
f24ae1d79fe15839495d7e813af3adbb	2007-01-09 16:13:33	2007-04-20 16:18:23	408
2b41d62f20225b65d618fa9e4270e564	2007-04-20 15:01:42	2007-04-20 15:59:36	2
5d2cbf4a3842b10cee259ec385ec5d4c	2006-12-20 11:28:33	2007-04-20 15:39:18	25
87df95881b74ce59ff086768f6b69a7a	2007-04-20 14:44:31	2007-04-20 14:44:31	1
b3e70a483ad2b5f9e59919bbce0e69e5	2007-03-12 16:12:09	2007-04-20 10:42:39	3
edc2eceec44db98c53baa6dfd797b151	2006-10-12 23:06:13	2007-04-20 01:32:26	2576
5b51aec1c7b2b7ebb2db6406556099c6	2007-02-15 21:53:31	2007-04-20 00:57:40	1140
ef705ef3e6259c5157e10be406d950e9	2007-03-01 10:55:51	2007-04-19 21:30:29	265
5f1985d811c5161925e26e59b3ae8d43	2006-11-29 17:21:41	2007-04-19 20:05:05	20
98f1b1e45875c788b7d23bec48bf0c17	2007-03-29 14:16:15	2007-04-19 19:26:51	45
865915650a85e7c27cdd11850a13f86e	2006-10-25 20:30:03	2007-04-19 03:36:16	18
044bfda21be6cb1f85f4bccc8e59c2e2	2007-04-13 11:39:02	2007-04-19 01:38:50	2
2c29e26f2ea6ef179c085049f8a89228	2007-01-20 23:34:13	2007-04-18 23:20:29	6
d1ee9d2e39e06495aa8b457e3d2d75a4	2006-04-28 16:39:24	2007-04-18 21:15:53	21
2b51f2a8f8b6bb611acfb541a0c11290	2007-04-18 12:47:31	2007-04-18 12:47:31	1
cc254540252410cc34eeaa640aaaced9	2007-03-15 16:48:42	2007-04-18 06:03:05	5
3a0a745ebb2bf12ef8a76b2ac34d8a4c	2007-04-10 19:31:23	2007-04-17 20:27:23	4
5f0893d05e1b8c82994405df4d472230	2007-04-17 17:32:51	2007-04-17 17:32:51	1
7952fec43a17888d3d457ed19f2664c3	2007-03-26 05:29:30	2007-04-17 14:55:44	66
8610d84bd092753aacaffef910a5891e	2007-04-15 19:13:11	2007-04-17 01:14:26	2
d2e34cccc9ae6f44f10aa556445503fc	2007-04-16 21:22:48	2007-04-16 21:22:48	1
d9d81e1c35ef67af7279d1d16e1d9327	2007-04-14 11:50:03	2007-04-16 20:42:07	18
e9f8b8b68e139f0cdf7f73690a320347	2007-04-06 01:17:48	2007-04-16 19:55:06	10
87339013ea09c24650b5c4cf9e4dbc7b	2007-03-06 20:54:28	2007-04-16 19:21:03	49
015ee31357dc567c3b70475d1f76132a	2007-04-01 16:48:07	2007-04-16 15:37:50	66

Figure 7: Binaries Downloaded

3.3.2.2 Bot interaction

Observing the interactions of the bots that matriculated through our architecture yielded very interesting results. Ports were opened, processes shutdown and/or restarted and new registry keys created. The malware usually restarts legitimate Windows processes so that it may append itself to that process. For example, msmgs.exe is the MSN Messenger process and, by default, is loaded on startup causing the malware to be reloaded every time the machine is restarted. In a high number of instances, the malware (hardens) the system to prevent other bots from infecting the machine with any further attacks and leaves the system still accessible, so that the casual user would not notice much difference. Only a small number of times has a piece of malware completely disabled the image causing it to be unusable. To use a concrete example, the malware Trojan.Mybot-7663 initially loaded the files lssas.exe and fswinsys.exe, which are registered as the W32.AGOBOT.RL Trojan and Worm.Ircbot.Gen respectively. It furthered its assault by unloading 90 drivers from memory including cdrom.sys, ultimately rendering the CDROM useless. It proceeded to unload 250 DLL files and deleted 77 services, most notably the secondary logon service causing major problems in logging into the image. After opening a few select ports, the malware terminated 16 processes, many system critical. These processes included lsass.exe, winlogon.exe, and services.exe and even though all were eventually restarted it is safe to assume that they were tampered with. All of the malware that we have actively examined use some type of systematic scan, presumably for propagation. Most of these were TCP SYN scans on a class B subnet. If a TCP SYN scan was not used, ICMP ping scans were used. We have noticed that DNS queries were hard coded into the bots, using the returned IP address to log into an IRC server and obtain secondary injections. Some malware ran had been relatively inactive until the completion of the secondary download in which a propagation scan would ensue. A high number of malware have displayed this behavior allowing us to form the hypothesis that malware writers use other writer code to ensure a small, compact binary. For example, our Nepenthes sensor captured a process called fswinsys.exe for the first time on May 10, 2006 and since have seen numerous hits per day. Upon execution, we realized that fswinsys.exe is able to initiate a propagation scan a lot more quickly than most other malware. After this realization we ran numerous other malware that would download the fswinsys.exe process as a secondary injection and used for propagation scans. This discovery lead us to our second hypothesis, which is that many of the malware writers use previously created malware or copy and paste code from previously created malware. For example, the malware following the md5sum 429d74b465003ddcfd54b586705191cb (classified as a W32.Spybot.Worm) displayed the above mentioned behavior. Its initial execution resulted in PCAP slices ranging from 200K to 600K. Once the secondary injection of fswinsys.exe was complete the next slice was 7.8M. The propagation scan had a time limit associated with it, so on completion the PCAP slices fell back to its 200K to 600K average. The malware then received a second propagation scan command the following day, but with no time limit and a longer delay resulting in PCAP slices ranging from 1.5M to 6.9M. This malware has become common among our analysis team in which the fswinsys.exe process is used to initiate large propagation scans. Malware often use IRC channels to receive commands for propagation scans and secondary download. Throughout the life of our Honeynet, these bots have shown an interesting similarity in the type of commands received. A main focal point for all malware is the use of a propagation scan. The common command for a propagation scan has been .advscan ¡port number¿ ¡threads¿ ¡delay¿ ¡time¿ ¡switches¿. For example, the command $.advscanlsass_4 4520050 - r\ -b\ -s$ would correspond to a randomized ($-r$ switch), class B ($-b$ switch) subnet scan on port 445 using 200 threads with a 5 second delay for an infinite amount of time. Rarely does a piece of malware designate a time for the scan to finish so the 0 is used to express an infinite amount of time. Furthermore, the $-s$ switch is a silent switch that bots will use to keep their status from being broadcast across the IRC channel.

3.3.2.3 Bot correlation

There was a variety of bots that were captured and analyzed using our format. Table 2 shows a list of some of the bots we captured.

Table 2: Bots Captured and Analyzed

Binary	Packer	ClamAv	NortonAv
44115764	tElock.98b	Worm.Vesser.A-1	W32.HLLW.Deadhat.B
04cb8	EXECryptor 2.2.4	Trojan.SdBot-2215	W32.Spybot.Worm
0c28c27	NsPack V3.3	Trojan.Mybot-5031	W32.Spybot.Worm
0ce21f	eXPressor v1.2	None	W32.Spybot.Worm
20d414	Morphine 1.4	Trojan.SdBot-1836	W32.Spybot.Worm
3d3525610	Unknown	Unknown	Unknown
429d7	EXECryptor 2.2.4	Trojan.SdBot-2275	W32.Spybot.Worm
4a94c	Packman V1.0	None	W32.IRCBot
54085f0	UPolyX v0.5	Trojan.Mybot-7706	W32.Virut.A
54d52	Unknown	Trojan.SdBot-2976	W32.Spybot.Worm
5525d6e6	Unknown	Unknown	Unknown
98c9d4aa	ASProtect 2.1	Trojan.Mybot-5151	W32.Linkbot.A
ab5335dab	Unknown	Unknown	Unknown
b48811	PEnguinCrypt 1.0	Trojan.SdBot-2564	W32.Spybot.Worm
c36dc	ASProtect 2.1	Trojan.Mybot-7669	W32.IRCBot
cd433	Unknown	Trojan.SdBot-4053	W32.Spybot.Worm
d8f04a	Unknown	Trojan.SdBot-4053	W32.Spybot.Worm
edc2e	ASPack	Trojan.SdBot-4053	W32.Spybot.Worm

Out of the 19 bots in the table several of them have the infected file lssas.exe. Table 2 also shows that several of the bots use the same packers. These similarities in the bots tells us that it is highly likely that these bots, although they come from different addresses geographically, have some of the same code in them. Overall we estimate that 70 percent of the bots in our repository come from the same bot source code. This supports the claims made by other publications [76, 12].

3.4 Conclusion

In this chapter, we have discussed our Honeynet-based Bot Analysis Architecture. We have shown that our testbed has been an invaluable learning tool for our students and allows them to directly observe interactions of malware in the wild. Our approach in this chapter has also provided us with the ground work necessary to discover characteristics within the network traffic of attackers that can lead to the reduction of attacks in the future.

CHAPTER 4: ATTACK ANALYSIS FRAMEWORK DEVELOPMENT

In the previous chapter we created a testbed that was able to capture botnet traffic. After capturing the bots we were also able to perform analysis on them to discover some interesting facts like how many of them are related to one another. One of the most important lessons we learned was that mechanisms that detect botnets based on signatures, such as firewalls and IDS, suffer from low detection rate when bot variants are introduced into the wild. This and other lessons learned from our honeynet testbed led us to develop our framework for botnet analysis. In this chapter, we introduce a network centric analysis framework to work towards detecting and preventing botnet attacks. For our framework we continue to use the components created in the Honeynet Based Architecture in the previous chapter. These components are the network monitoring system component, malware collection system component, closed analysis system component, open analysis system component, correlation system component, and the repository system component. In this chapter we make slight changes to the nomenclature of the components previously mentioned. The new names of the previously mentioned components are network monitoring component, bot capture component (because now we are only interested in malware that exhibits bot capabilities), closed analysis component, open analysis component, correlation repository component (we combined the correlation and repository components into one component since the correlator collects all the data from the other components). We also added the taxonomy component which organizes the analysis by bot, the threat component which discovers the threat each bot poses, and the protection framework component, which produces a report that displays the threats of the bots. In this chapter we also introduce two implementations. The first implementation is "Nepenthes +" which uses Ruby scripts to dynamically add vulnerability modules on the fly to capture more bots without a significant downtime. This is an implementation of the bot capture component. The second implementation is the Intrusion Analysis Forensics Tool (IAFT) which implements the correlation component by receiving various inputs and discovering correlation links between them. In this chapter we focus on learning more information about the bots by identifying malicious characteristics through the network traffic. Once we have their characteristics we can then decide what the threat level of each bot in the botnet is determined to be. The remainder of this chapter is organized as follows. Section 4.1 discusses background technologies and related work. The analysis framework is presented in Section 4.2. We then discuss the component development in section 4.3. In Section 4.4, we discuss our preliminary results and section 4.5 concludes the chapter.

4.1 Background

This section is a review of the background behind our bot analysis architecture. Some of this information is repeated from earlier discussion. Botnets continue to be a disruptive force in computer networks today [18, 54, 24, 40]. Honeynets have been used to learn as much about bots and the attacker sending bots as possible [75].

Even though this approach allows us to gather attackers footprints, a systematic data analysis method is still needed. In the botnet, the command and control is where the attacker sends commands to the botnet. Currently most malicious bots use IRC to communicate with the command and control. IRC built-in multi-cast capabilities make it easy for the commander to send orders to all the bots in the botnet without much effort [47]. A more destructive form of communication for bots is with the P2P protocol. These bots contain P2P clients and can communicate with one another without the use of a central command center. With this type of command and control the attacker can initiate commands by posing as a peer anywhere in the network. The power of P2P botnets was discussed in [39]. Other forms of command and control are also being used to a lesser degree, such as instant messaging and cellular phones. As researchers continue to find ways to protect against IRC based command and control structures, the number of botnets controlled by other protocols will continue to increase. As mentioned earlier, DDoS attacks are extremely difficult to detect. Most existing mechanisms have limitations to properly distinguish botnet traffic from legitimate traffic, generating a high false positive rate [23]. A high false positive rate may be its own denial of service, since legitimate traffic is blocked from accessing the network. Botnets continue to be a growing threat until a trustworthy mechanism is presented that effectively detects and blocks botnet attacks while allowing a very low false positive rate [9, 10]. Defending networks against botnet attacks is an emerging issue in network security and cyber crime research communities. To our knowledge, there are a few works using risk as a deciding factor such as a newly released McAfee Advanced Botnet Protection in Intrusion Prevention System [8]. This tool takes a similar approach of our framework in that it uses a proxy to accept or block traffic that appears to be botnet related, but it does not use the risk value rigorously but mainly relies on a signature based approach.

4.2 Overview of our framework

This section gives detail of our approach that is based on three critical requirements we identified in the last chapter. These requirements are as follows:

1. **Bot Detection:** Systematically collect and analyze bot traffic over the Internet
2. **Bot Characteristics:** Comprehensively discover characteristics and unique behaviors
3. **Bot Threats:** Dynamically determine threats and generate protection reports

Our analysis framework consists of several components to satisfy these requirements:

4.2.1 Bot detection

Bot detection involves identifying both known and unknown bots that are trying to enter the protected network. We accomplish this by installing a malware collection system component on the network. **Malware collection component.** The type of bots collected on each network are different because attackers target certain subnets with different bots. The malware collection system component uses emulated vulnerabilities to entice attackers and to trick them into believing they are interacting with the actual vulnerable services. The attackers send their bots to this system and it is captured and not run. To realize this architecture we built upon a tool call Nepenthes [56]. Nepenthes has to restart each time a new module is created for a vulnerability. This is not feasible for us, since we want to start capturing bots as soon as a new vulnerability is found. To correct this discrepancy, we use a Ruby program to provide script space where modules are added on the fly and the system does not have to restart to go into effect.

As new vulnerabilities are found in software, modules to capture bots that target these vulnerabilities will be added to the collection system. In addition to the vulnerability modules to capture bots, is a scanning module to capture the scanning activity produced before the bot is downloaded to the collection system. This enables us to know the full story of each compromise, which helps us to develop concise characteristics for blocking the bots in the future.

4.2.2 Bot characteristics

To comprehensively discover characteristics and the unique behavior of bots, the system is required to identify known malware, discover new malware, discover traffic patterns of individual malware, and discover a correlation between more than one instance of malware. **Network monitoring.** The network monitoring component is employed to identify known malware signatures. In this system, both an antivirus and a firewall analyze the traffic to discover if anything matches their current rule set. The discovery of new malware is done by detecting the variations in the traffic using the malware collection system component. Once the traffic has been determined to have malicious packets, it will update the vulnerability list in the component and create a new module to capture the malware. To discover the traffic patterns, the bot is then automatically sent to the **closed analysis component** where it is ran in a virtual network and analyzed to return characteristics based on the behaviors it showed in the analysis. The initial characteristics obtained using the closed analysis will include the strings and PCAP packets from simulated attacks. The **open analysis system component** is then used to run the malware on the Internet. All transactions to and from the network are monitored and blocked if the malware we have installed is a significant contributor to an attack. The characteristics discovered here are the actual characteristics of the communication between the malware and the attacker. The malware characteristics are then correlated and displayed using the **correlation system component**. This system provides the ability to display the intelligence discovered from the characteristics. All alerts from the **network monitoring system component** are displayed in the intelligence report. This report is generated by querying the open and closed analysis system components and connecting the characteristics found by their md5 value, which is discovered in the **malware collection system component**. The **correlation system component** uses keywords found using the strings command from the **closed analysis system component** to search the Internet for possible characteristics, such as motives or frequency of meta-data of the attacker. This can include IRC commands, hard-coded DNS entries, usernames, and so on. These characteristics are then added to the data from the open and closed analysis systems in the correlator. The **correlation system component** also finds links between different malware to discover patterns that may exist between different types of malware. The **taxonomy system component** keeps a record of all bot characteristics. All characteristics in the taxonomy are able to be correlated. When a relation is found that connects multiple characteristics of malware, the **correlation system component** will send an update to the **taxonomy system component** which makes a reference to the malware that has been correlated.

4.2.3 Bot threats

An examination of the system is taken to discover what the vulnerabilities are. This examination takes into account the applications installed, the operating system and the importance of their vulnerabilities to the mission of the network. The more important an application or service is to a network, the more weight the vulnerability carries as it pertains to the threat-value. As we identify evidence of targeting these vulnerabilities in increasing the threat-value of that particular IP traffic, subsequent packets are recorded in a suspicious traffic storage component and not allowed to access the network until a certain level of legitimate traffic is recorded in the proxy. This is dynamic since every window of traffic would be different and increase and decrease the risk-level on the fly. All the components work together to discover as much information about the malware as possible to be able to discover whether or not they present a threat to the network. The threat engine is unique for each particular system it is protecting because risk is relative to the individual, or company that is being protected. Each entity has its own risk factors and different weights for each factor. The first step in this engine is to discover the personalized risks. These risks include the vulnerabilities of the operating systems being used on the protected network, the applications, the known vulnerable services on the network, and the importance of the services. After the personalized risks are detected, we receive input from the Taxonomy System Component to give the malicious characteristics and then use the combination of the personalized risks and the characteristics to give a risk value that is used to aid in the determination of whether traffic is blocked or not. We implement the threat analysis section in the next chapter.

4.3 Component development

In this section we discuss the development of the components that satisfy the three requirements discussed earlier.

4.3.1 Bot Detection Development

To capture bots, Nepenthes uses vulnerability modules which are loaded at run time. These modules emulate vulnerabilities that are currently found and exploited in the wild. The following is partial code of the lsass vulnerability module:

```
Nepenthes *g_Nepenthes;

/**
* The Constructor
* creates a new LSASSVuln Module,
* LSASSVuln is an example for binding a socket
LSASSVuln::LSASSVuln(Nepenthes *nepenthes)
m_ModuleName = "vuln-lsass2";
m_ModuleDescription = "modules provides lsass emulation";
m_ModuleRevision = "$Rev$";
m_Nepenthes = nepenthes;
m_DialogueFactoryName = "LSASSDialogue Factory";
m_DialogueFactoryDescription = "creates dialogs to emulate lsass";
```

In order to add new vulnerability modules Nepenthes has to completely restart which could take a significant amount of time depending on the complexity and the number of modules that were being added to capture bots. To alleviate this scenario we developed a new version of Nepenthes we call ”Nepenthes +”. ”Nepenthes +” uses ruby scripts to load the vulnerability modules on the fly. Figure 8 is a diagram of our ”Nepenthes +” implementation.

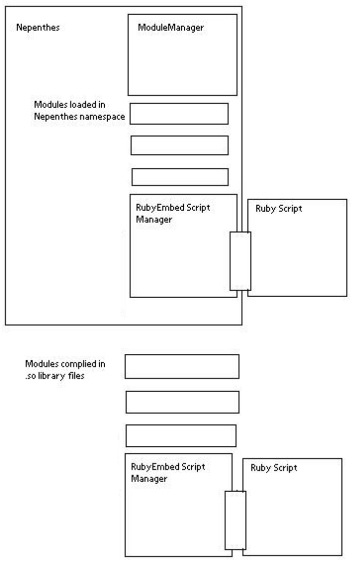
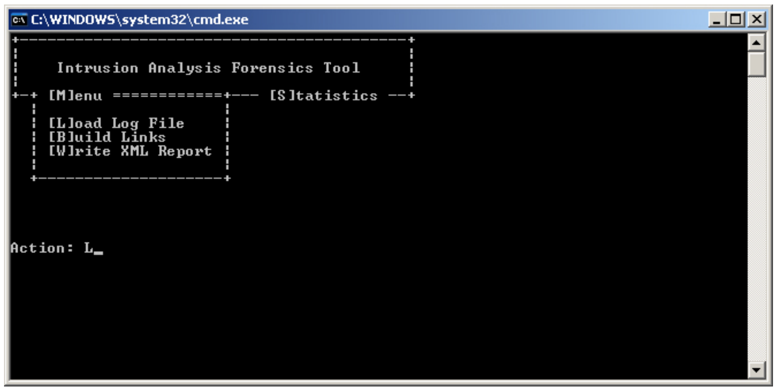


Figure 8: Nepenthes + Diagram



```
C:\WINDOWS\system32\cmd.exe
+------------------------------------------+
|                                          |
|    Intrusion Analysis Forensics Tool     |
|                                          |
+-+ [M]enu ===========+--- [S]tatistics --+
  |                   |
  | [L]oad Log File   |
  | [B]uild Links     |
  | [W]rite XML Report|
  |                   |
  +-------------------+


Action: L_
```

Figure 9: IAFT Menu

4.3.2 Correlation Development

We implemented the correlation component by creating a Perl based program we call the Intrusion Analysis Forensics Tool (IAFT) that takes in multiple files as input, builds links between the files, and then writes an XML report. Figure 9 shows the menu of the correlation component.

4.3.3 Taxonomy Development

The taxonomy is a comprehensive collection of each individual bot characteristic based on the bot. The purpose of the taxonomy is to determine to what extent the bots that come across our network are related. We implement the taxonomy component using MYSQL for the storage and Perl scripts to update the database with new relations.

4.4 Preliminary Results

Our results demonstrate how our framework can help us identify different classes of bots. Here we show the results from the bot detection component, the correlation component, and the taxonomy component.

4.4.1 Bot capture results

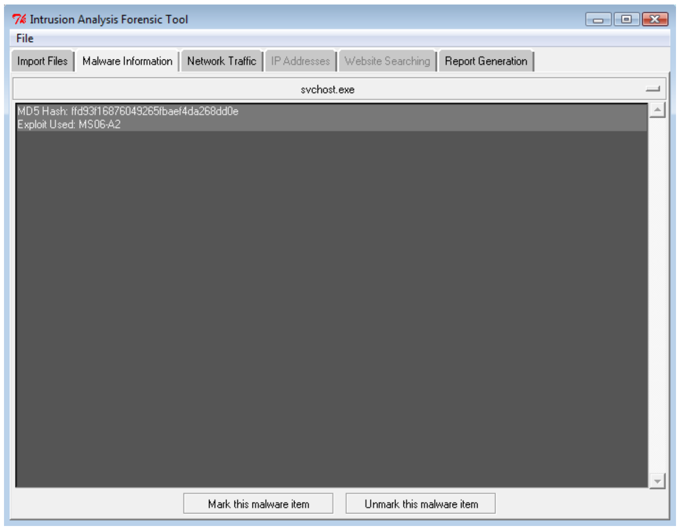
Table 3 illustrates a list of bots that we examined and analysis results of each bot. Each bot has an identification based on MD5 value. Also, we identified whether the collected bots are known or unknown bots. The targeted vulnerabilities were captured and further interactions with our system were also monitored. Those interactions include system changes, DNS queries and IRC communication, network service, and IRC communications with the intent.

Table 3: Comparison of Bots Ran Through the Analysis Process

Bot	Ident.: MD5	Vuln. Targeted	DNS	IRC Comm.
Unknown	3d35 ..	ms04-011	bacho.us	Yes: Checks paypal
Mybot-7706	0c28 ..	ms04-011	bacho.us	Yes: Checks paypal
Unknown	5525	ms04-011	asechka.ru	Yes: Reptile Welcomes
Mybot-7669	C36d ..	ms04-011	prison.net	Yes: Reptile Welcomes

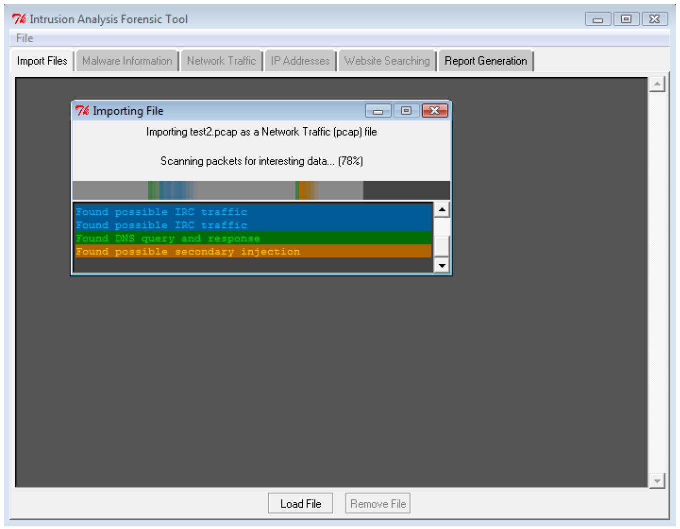
4.4.2 Bot correlation results

We created the IAFT to implement our correlation component.



```
MD5 Hash: ffd93f16876049265fbaef4da268dd0e
Exploit Used: MS06-A2
```

Figure 10: Bot Uploaded Into Correlator



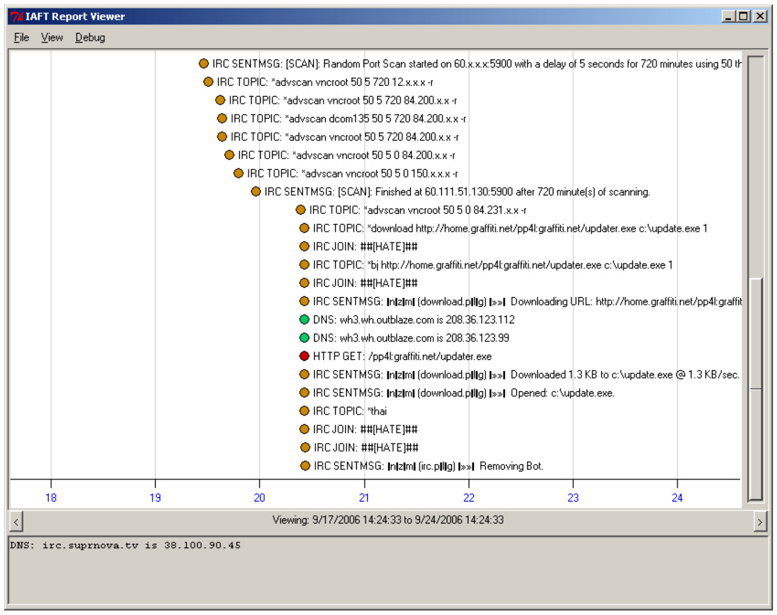
```
Importing test2.pcap as a Network Traffic (pcap) file

Scanning packets for interesting data... (78%)

Found possible IRC traffic
Found possible IRC traffic
Found DNS query and response
Found possible secondary injection
```

Figure 11: PCAP Uploaded Into Correlator

Figures 10 and 11 show the bot capture file being uploaded using our Intrusion Analysis system. In figure 10, the binary is uploaded from our closed analysis component and in figure 11, the PCAP file is uploaded from open analysis. Once the analysis is completed, each analysis item is stored in our bot taxonomy to categorize the bot for articulating patterns of bot centric attacks. Using the open and closed analysis results, we also developed a correlation report so that we can identify all relevant system and network activities performed by a particular bot. As shown in Figure 12, more fine-grained intelligence report for the bot can be generated. Also, it allows us to further examine network and system activities at the specific time frame during the course of investigation actions.



```
IRC SENTMSG: [SCAN]: Random Port Scan started on 60.x.x.x:5900 with a delay of 5 seconds for 720 minutes using 50 th
IRC TOPIC: *advscan vncroot 50 5 720 12.x.x.x -r
IRC TOPIC: *advscan vncroot 50 5 720 84.200.x.x -r
IRC TOPIC: *advscan dcom135 50 5 720 84.200.x.x -r
IRC TOPIC: *advscan vncroot 50 5 720 84.200.x.x -r
IRC TOPIC: *advscan vncroot 50 5 0 84.200.x.x -r
IRC TOPIC: *advscan vncroot 50 5 0 150.x.x.x -r
IRC SENTMSG: [SCAN]: Finished at 60.111.51.130:5900 after 720 minute(s) of scanning.
IRC TOPIC: *advscan vncroot 50 5 0 84.231.x.x -r
IRC TOPIC: *download http://home.graffiti.net/pp4l:graffiti.net/updater.exe c:\update.exe 1
IRC JOIN: ##[HATE]##
IRC TOPIC: *bj http://home.graffiti.net/pp4l:graffiti.net/updater.exe c:\update.exe 1
IRC JOIN: ##[HATE]##
IRC SENTMSG: |n|z|m| (download.p||g) |»»| Downloading URL: http://home.graffiti.net/pp4l:graffit
DNS: wh3.wh.outblaze.com is 208.36.123.112
DNS: wh3.wh.outblaze.com is 208.36.123.99
HTTP GET: /pp4l:graffiti.net/updater.exe
IRC SENTMSG: |n|z|m| (download.p||g) |»»| Downloaded 1.3 KB to c:\update.exe @ 1.3 KB/sec.
IRC SENTMSG: |n|z|m| (download.p||g) |»»| Opened: c:\update.exe.
IRC TOPIC: *thai
IRC JOIN: ##[HATE]##
IRC JOIN: ##[HATE]##
IRC SENTMSG: |n|z|m| (irc.p||g) |»»| Removing Bot.
```

Viewing: 9/17/2006 14:24:33 to 9/24/2006 14:24:33

```
DNS: irc.suprnova.tv is 38.100.90.45
```

Figure 12: Correlation Report

4.4.3 Taxonomy results

Table 4 shows an abbreviated taxonomy of bots. Here each bot is searched for relations to new bots entering the taxonomy. When a relation is found it is listed under both bots that are related. Our results show that 0c28 .. is related to the three other bots in the table 4 because of the vulnerability that was targeted. This is

Table 4: Abbreviated Taxonomy of Bot 0c28

Bot	Trojan.Mybot-7706/ W32.virut
Identification: MD5	0c28 ..
Vulnerabilities Targeted	ms04-011
System Interaction	msnserve.exe
DNS Queries	bacho.hassouna.us
IRC Communications	Yes: Checks for paypal account
Bot Relation	3d35: vulnerabilities targeted (ms04-011)
Bot Relation	5525 : vulnerabilities targeted (ms04-011)
Bot Relation	C36d: vulnerabilities targeted (ms04-011)

typical of the other bots that were captured on our system which strongly suggests that most bots are related and come from a small pool of source code.

4.5 Conclusion

In this chapter, we have introduced a network-centric attack detection and prevention analysis framework. We discussed the network analysis component and demonstrated how we could implement the correlation component. Also, we described functionalities and features of each component in the framework. In addition, we demonstrated the feasibility of our framework based on the earlier mentioned testbed.

CHAPTER 5: FRAMEWORK DEVELOPMENT WITH IRC SANDMAN

The framework introduced in the last chapter was a general framework which allows for any method to be inserted as long as the method satisfies the rules of each component. Based on the rules of our framework, this chapter focuses on our analysis method of IRC bots using an IRC monitoring tool called IRC Sandman that observes IRC traffic and automatically downloads secondary injections from a command and control center. Using our framework we also identify characteristics and behaviors of network-centric attacks focusing on malware-based bot attacks. Such characteristics can be used to determine relevant threat values of specific network patterns for making signature-based detection more effective. Particularly, we focus on how to use IRC Sandman to analyze IRC bots and how to monitor real-time conversations in IRC channels that would eventually lead us to have more meaningful characteristics of bots. In other words, our goal is to identify bot characteristics and to detect botnet traffic based on the threat level derived from the identified characteristics. Each requirement is related to each other and is designed to work together to meet the overall goal. The rest of this chapter is organized as follows. Our framework and its realization are presented in Section 5.1. In Section 5.2, we discuss the analysis methods used. The implementation of IRC Sandman is discussed in section 5.3 and the results are discussed in 5.4. Section 5.5 concludes this chapter.

5.1 Overview of Our Framework and Its Realization

This section gives an overview of our approach that is based on three critical requirements as follows:

1. **Systematically collect and analyze bot traffic over the Internet (Bot Detection)**
2. **Comprehensively discover characteristics and unique behaviors of bots (Bot Characteristics)**
3. **Dynamically determine threats and generate protection reports (Bot Threats)**

Our framework consists of several components that satisfy these three requirements which we will refer to as **Bot Detection, Bot Characteristics, and Bot Threats**. The **Bot Detection** requirement defines any front-end component that will interact with the Internet. This component is used to sense and record network traffic and to articulate `bot characteristics` present in a window of network traffic. Each recorded packet is examined and characteristics are extracted from them. The packets are grouped into windows that are sent to components that implement the **Bot Characteristics** requirement. Each window is from a separate IP address, so there are multiple windows being recorded at one time. A second function of components in the bot detection requirement is to compare recorded windows. If the windows show additional characteristics of a botnet attack, the threat levels of each of the source IPs the window originated from automatically increase past the threshold level. These mechanisms will ensure that botnet attacks are identified and blocked after a small amount of traffic is allowed through the architecture and to the network. In the **Bot Threat** requirement, the characteristics discovered in the packets are given a threat level based on the importance of each characteristic and are then cross checked with possible vulnerabilities the traffic could exploit. Each of the vulnerabilities is given a mission level that describes the level of importance based on the services provided. Components that satisfy the bot threat requirement also compute a threat-level and generates a corresponding protection report. When system protection is implemented, our approach is dynamic in that the characteristics and values used to make a decision to block or pass traffic need to be done automatically so they can effectively protect a system. The threat engine within a component of the bot threat requirement leverages unique features of each particular protected system due to the fact that threat values are relative to the company or entity that is being protected. Each entity will have its own risk factors and different weights for each factor. The first step in implementing this engine is to discover the personalized risks. These risks include the operating systems being used on the protected network, the applications, the known vulnerable services on the network, and the importance of the services. Based on the proposed framework, we initially identified four components to implement the tasks of the requirements. Each component can also act as a stand-alone mechanism to aid in bot analysis. Figure 13 shows how the proposed framework is realized in our testbed architecture. The solid line represents network traffic generated that is not originating from the system components. The dotted lines represent traffic that is generated from the system components. We briefly describe these components and related functionalities.

- **Malware Collection and Network Monitoring:** Before we can discover what the risks are in a network, we need to discover how attack code reacts with the system. To realize this goal, a collection system is proposed that collects bots to be dynamically analyzed. Dynamic analysis occurs by ensuring that the collection system emulates each of the services on the network it is

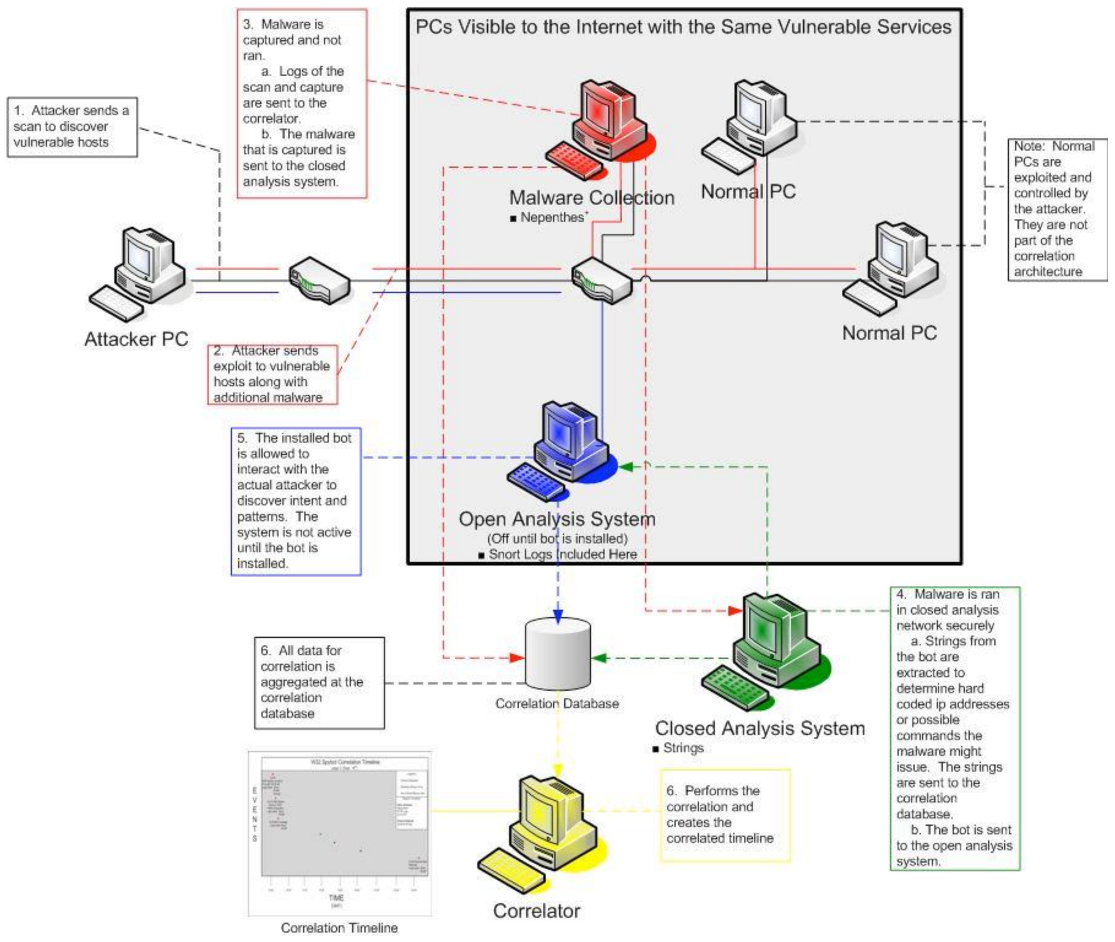

PCs Visible to the Internet with the Same Vulnerable Services
1. Attacker sends a scan to discover vulnerable hosts
3. Malware is captured and not ran.
a. Logs of the scan and capture are sent to the correlator.
b. The malware that is captured is sent to the closed analysis system.
Malware Collection
■ Nepenthes+
Normal PC
Note: Normal PCs are exploited and controlled by the attacker. They are not part of the correlation architecture
Attacker PC
Normal PC
2. Attacker sends exploit to vulnerable hosts along with additional malware
5. The installed bot is allowed to interact with the actual attacker to discover intent and patterns. The system is not active until the bot is installed.
Open Analysis System
(Off until bot is installed)
■ Snort Logs Included Here
4. Malware is ran in closed analysis network securely
a. Strings from the bot are extracted to determine hard coded ip addresses or possible commands the malware might issue. The strings are sent to the correlation database.
b. The bot is sent to the open analysis system.
6. All data for correlation is aggregated at the correlation database
Correlation Database
Closed Analysis System
■ Strings
6. Performs the correlation and creates the correlated timeline
Correlation Timeline
Correlator

Figure 13: Honeynet Based Architecture

protecting. We capture bots by emulating the vulnerable services attackers target and downloading but not executing the bot. Initially the collection system had to restart every time a new emulated module was added to the system. To alleviate that problem we created "Nepenthes +" which added modules on the fly [2]. Also, this system provides protection against significant involvement in attacks after the bot has been ran on the system. It uses firewall and intrusion protection techniques, such as limiting or dropping packets leaving the protected network.

- **Analysis System Components:** This component takes the binary captured in the collection system and runs it on a closed network environment. This is a necessary step to discover certain aspects of the malware before putting it on the open analysis system and opening it up to the network. The closed analysis component has the capability to use attack commands found in the binary and perform simulated attacks using a Perl script. These attacks are only run in the simulated network and will give insight to what the binary is made to be used

[2]details in previous chapter

for. It includes the discovered hard-coded DNS addresses, attack commands, and other functionality of the bot. Eventually more functionality will be used from the closed analysis such as patterns from the virtually simulated attacks that can be performed within the closed analysis system. The open analysis component of this system involves injecting a computer with a malicious bot and allowing it to connect back to its original destination. This allows us to isolate the bot and monitor its traffic in a more controlled way instead of waiting to be infected and then monitoring the traffic.

- **Pattern Correlation System Components:** The pattern correlation system takes input from the open analysis and closed analysis systems and creates an intelligence report to display the alert events that are identified from the bot installed. This intelligence report is used to discover patterns in the traffic and correlations between logs. The goal of the correlation system is to gather as much information (characteristics) about each individual bot as possible and correlate the results with other bots to discover a taxonomy of each bot. As mentioned above the patterns discovered using the correlation system for each particular bot is stored to the taxonomy. Each bot taxonomy will have a list of its own characteristics as well as references to other bots that use or have any connection with the bot entry in the taxonomy. This information is referenced by the engine. The purpose of the taxonomy is to provide a comprehensive identity for the bot so the characteristics provided by the identity will lead to an accurate assessment of the risks they present. A taxonomy updater is also needed to keep the taxonomy up to date and accurate. When a new correlation is found in a bot, its taxonomy entry will change to reflect the new correlation. The repository is a central collection of all the logs in our architecture. This gives the administrator a macro view of the protection system and provides an aggregated view of the attackers on the network. The repository holds statistics and geographical information on the logs and presents them as input to the engine to be used as a factor in the assignment of risk to the traffic.

5.2 Understanding Bots

The previous section gave an overview of the current system components in our framework. In this section, we go into more detail of how we have analyzed the bots in our testbed architecture and what we have learned from our analysis.

5.2.1 Analysis method

As mentioned earlier, the analysis method used in our architecture involves capturing a bot and analyzing it both on and off line. Malware collection is achieved using our modified version of Nepenthes [6] which we call "Nepenthes +", that emulates Microsoft Windows services to incite automated attacks. When an attack occurs, "Nepenthes +" logs the malicious activities and attempts to download any binaries associated with the attack. The downloaded malware is automatically stored in a MySQL database, as well as the originating IP address, and run through two anti-virus engines, Norton 10 Corporate and ClamAV. The anti-virus engine results are then stored in the database. An important part of understanding botnet interaction and propagation commands is through service emulation. By creating services that act like real services (such as an FTP server that speaks the protocol but denies any requests after all protocol negotiation), it becomes easier to record and understand how bots communicate, propagate, and receive commands from the botmasters. An example of a product that performs service emulation as well as recording network traffic and creating memory dumps of malware program space in an automated context is the Truman Sandnet [89]. The sandnet uses a client-server architecture by running the malware on the client machine, emulating services on the server, and collecting data from the client machine and the network communications with the client. Sandnet provides an isolated environment and a virtual network for the piece of malware to execute. The environment consists of two computers, a Sandnet Server and Sandnet Client. After the initial execution of malware, an md5sum file, memory dump file and network traffic logs are sent to the Sandnet Server from the Sandnet Client. Using a specifically designed Perl script we can recompile the memory dump file and running the Linux command (strings /minus a ¡file¿) to obtain the strings off of the malware. The strings allow us to determine commands used by the malware as well as target areas that the malware will be likely to hit. Furthermore, Sandnet is able to simulate various types of servers, the most important being an IRC server since this is the most notorious avenue for sending malware commands. This Sandnet-based analysis is all off-line. The Sandnet is not connected to the Internet, so (production) data of the malware in its natural environment cannot be gathered. The Sandnet can also be used to classify malware and extracting IRC-based bots through interaction with the Sandnet simulated IRC server. The live execution environment is notably more verbose than using Sandnet. To begin, VMWare workstation is used to create a default installation of Windows XP, Service Pack 1. After the image is created, Sebek [74], is installed onto the image. Sebek is a kernel based data capturing tool and captures the processes used by the image, sending them as packets across the network. To obtain the files added, deleted, and changed by the malware the tool Perileyez [73], is run on the image. The initial snapshot of the image is taken once Sebek and Perileyez are installed and after the malware is executed, a second snapshot is taken. By comparing the two snapshots we can identify alterations the malware makes to the image including changes to drivers, DLLs, processes, ports and remote connections as well as any files changed. Capturing and analyzing network traffic is the final step in running a live execution environment. To capture all network traffic generated by the virtual environment, we use a Honeywall. The Honeywall is able to capture all network packets that are sent and received by the image. These packets are merged into PCAP files and sent to a central server at the end of each day. Currently we have found it useful to separate the PCAP files into four hour segments, giving us six slices for each day. By segmenting the file, it allows us to locate suspicious data more easily. Using the tool Wireshark [100], we can look at the daily PCAP files and determine the actions of the malware for the previous day. One PCAP file can display IRC conversations, secondary injections attempts, DNS queries, propagation scans and HTTP conversations as well as any other type of network traffic.

5.3 IRC Sandman

A common way for bots to communicate with botmasters is use of the IRC protocol [76], a plain text communication protocol used for hosting and managing chat rooms. Botmasters use IRC to send commands to the individual botnet nodes. Some common commands given to bots instruct the bot to perform vulnerability scans on the network, download and run a certain binary, send gathered data to a specific address, leave the IRC channel, or even uninstall itself and exit. Common methods for sending commands to a bot are to embed them in the IRC Channel topic or send them as a broadcast message to all clients on the channel. These commands can notify the bot to begin a port scan on a particular address range, perform an attack on a particular host, or download and run a binary from a particular host, such as a URL or FTP host. After months of research using honeypots as a reactive way of gathering malware binaries and using the Sandnet as a method of off-line automated analysis, analysis of IRC traffic was still being manually performed by capturing and parsing network traffic logs. Due to limited resources for analyzing botnets, malware is only run for a short amount of time before moving to another binary. It becomes difficult to continue monitoring IRC traffic, possibly missing important research data.

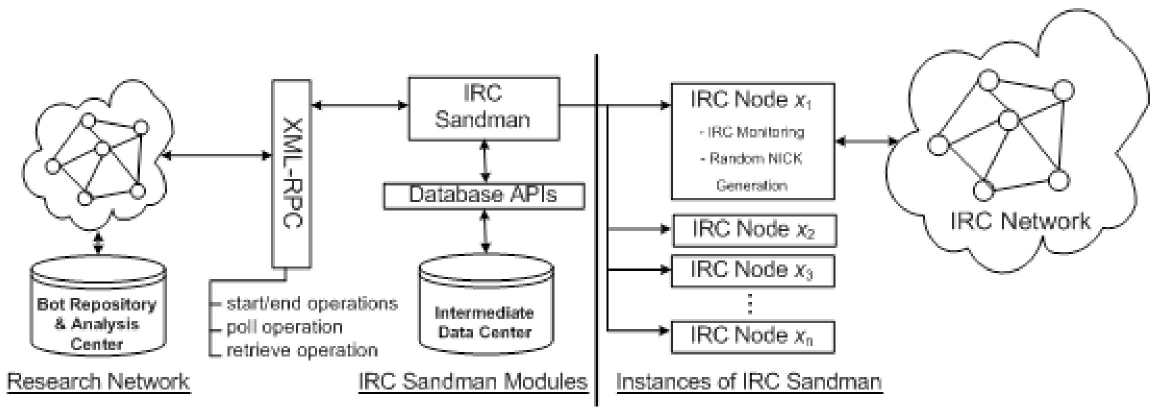


Figure 14: IRC Sandman Framework

IRC-based botnets often have a long lifetime that may require monitoring that is expensive to manually perform. The recent work by Rajab et al. indicated that roughly 84% of the IRC servers were active and 55% of them were continuously performing scanning activities over the three months of their study [2]. This demonstrates a need for an automated tool, scalable to many current research architectures that can monitor and perform minor analysis on these IRC channels. To address this problem, we introduce IRC Sandman 14, which is a custom analysis tool, written in Perl, for the purposes of monitoring known hostile IRC channels, often used in conjunction with botnets. Computers infected with bots will often log on and monitor an IRC channel for the purposes of receiving commands such as port scanning and DDoS attacks. In addition to attack commands, IRC channels can be used to instruct a bot to download and run another program from a specific location. The IRC Sandman has also been designed to download these secondary injections and store them for later use. The IRC Sandman aids in malware analysis by helping researchers understand the structure of botnets, methods that are used for secondary injection, and in some cases the psychology of botnet administrators and their levels of interaction. Tracking of botnet channel administrators can be performed across multiple IRC channels and servers to create botnet correlation and collaboration. The IRC Sandman consists of three primary sections as illustrated in Figure 14. The IRC Sandman body (Sandman driver), IRC Nodes (operated and administered by the IRC Sandman Body), and a Research network. Upon launching the IRC Sandman master process, a query is performed from the data center to gather a list of known, running IRC channels and data previously gathered for each. It then launches an IRC Node for each running IRC channel, which logs all traffic, downloads and stores secondary injections, and other tasks. Upon an IRC channel closing, the IRC Sandman being kicked from an IRC channel, or closing the IRC Sandman, all data obtained for that IRC channel will be uploaded to the data center in the research network. Construction of the data center, being a database or file system space, consists of two main parts: IRC Channel data/credentials and data gathered by the IRC Sandman system. The gathered data consists of logs obtained while monitoring IRC channels, binary dumps of secondary injections. The IRC Channel data consists of relevant server and channel information required to successfully login. This includes the IRC server host and port, username and nick, channel and channel key, channel modes to set, and miscellaneous data. The details of each component are as follows:

5.3.1 IRC nodes

The IRC Nodes perform the actual IRC interaction and logging. The IRC Nodes begin by connecting to the IRC server and logging in with a given USER and NICK, waiting for a 004 or 005 code from the server to indicate login success. If a 433 code is returned, indicating that the NICK is currently in use, a new NICK is generated similar to the given NICK. Once the IRC Node has logged into the IRC server, it attempts to join the specific channel that the botnet currently uses for establishing the communication. In the event of a channel redirect, in which the botnet communication has been moved to another channel, the IRC Node will change to the specified channel. All traffic sent to and received from the IRC server is logged to a file, with associated timestamps. This can include botnet commands, channel operators and their communications, private messages to all or a subset of bots, as well as some bot responses on the channel. A verbose mode is available that will also log communications and status information to the console. This allows an even more active monitoring of specific IRC channels when required. Also, Bots commonly use the NICK as a way of advertising the environment that it is running on. For example, the NICK: [00—USA—XP—780025] shows that the bot is on a Windows XP machine, located in the United States (USA). To create a unique NICK, bots often generate and append random numbers. The IRC Sandman takes an optional value for a number of digits to use in creating a NICK. This helps the IRC mask its identity as one that a bot would use. In the event of a (NICK already in use) error, a different but similar one can be generated and used. In addition to logging traffic, the IRC Sandman system is designed to automatically parse out and download binaries. Currently supported download methods are HTTP, FTP, and SFTP. In downloading these binaries given directly to bots in the wild, it is possible to obtain new malware and new bots before they are propagated to research honeypots. It is also possible to acquire binaries before companies such as Symantec and McAfee obtain them, which can give researchers a competitive edge in understanding new techniques and developing countermeasures to combat newly released malware.

5.3.2 IRC sandman server

The IRC Sandman Server acts as a gateway between the client research network that is implementing the application and the IRC Nodes performing the IRC operations. It listens to the client research network for XML-RPC commands specified later in this paper. The IRC Sandman Server is also responsible for the creation, destruction, and management of IRC Nodes.

5.3.3 XML-RPC operations

In order to cause minimal changes to research architectures upon implementation, the IRC Sandman was designed to communicate with client research networks via XML-RPC. These operations include start, end, poll and retrieve: start operation creates an IRC Node to listen to the specified IRC channel. Parameters include: IRC server host name, port to connect to, IRC parameters (NICK, USER, USERHOST, PASSWORD, CHANNEL, KEY, MODES), and random digits for random NICK generation described later in the IRC Nodes section; poll operation verifies that the IRC Sandman is currently listening to a given IRC channel. The parameters are the IRC server host name and the port that it should be connected to. The IRC Sandman verifies its listening status to the specified IRC server, and return true/false accordingly; end operation stops listening to the specified IRC server. The parameters are the IRC server host name and the port that it should connect to. The IRC Sandman scans its collection of listening IRC Nodes and gracefully stop the specified server if it exists. This saves any unsaved data; and retrieve operation gets a compressed archive of all files associated with the given IRC channel. The parameters are the IRC server host name and the port it should be connected to. The IRC Sandman scans its persistent data storage for the specified IRC server. If found, it creates a compressed archive of the log files and all downloaded binaries associated with the IRC server. The archive is then sent to the requester.

5.3.4 Client research network

Clients implementing the IRC Sandman have little to adapt to the integration of this system. Any existing system that is desired to access directly to the IRC Sandman requires an adapter to communicate with the IRC Sandman via the XML-RPC operations. All interpretation and manipulation of data gathered by the IRC Sandman is the responsibility of the implementing network. An implementing researcher can extract data specific to their research goals from the compressed archive given upon retrieval. Binaries can be sent to automated analysis systems to be run. Log files can be scanned as a method of enumerating and understanding command syntax or correlating botnet administrators across sites and botnets. Implementation from the client side is very open-ended, to provide a broad collection of potential research data to researchers. Since the IRC Sandman and the client research network communicate in plain text over XML-RPC, security for the data being transmitted is the responsibility of the implementing research network. This can be achieved by having communications isolated on a private network, or having a custom adapter on the same machine that communicates with other research network-specific applications over SSL if required. The IRC Sandman, in conjunction with other research tools, has lead to an increased understanding on botnet commands through correlation of command syntax. Functionality of these obtained commands can be studied by issuing the commands, often with experimental variation, to bots in simulated environments. Obtaining secondary injection immediately via a live IRC botnet command ensures that researchers are aware of secondary injections, which may be new versions of the bot or a new bot completely, and can be studied while they are still being deployed. This prevents research geared toward current trends from being compromised by self-propagating bots that may be old or not in use anymore. Also, in the event of a potentially disastrous attack, immediate study of a new injection may lead to earlier detection and prevention of attack methods. This showed the importance of the IRC Sandman tool which captures each secondary injection for further study. One such secondary injection capture is shown on the last line of Figure 3. Correlation of channel administrators (such as flyy and Albaboy[x] as illustrated in the Figure 3) with other channels assist researchers in correlating different botnets.

```
08:49:03 | :[XP]18655!~XP9326@----.0A6.5FB.IP JOIN :#!nja!
08:49:03 | :IRC.----.com 332 [XP]18655 #!nja! :#adcan asn1 4 0 -r
08:49:03 | :IRC.----.com 333 [XP]18932655 #!nja! flyy 1157499691
08:49:03 | :IRC.----.com 353 [XP]18932655 @ #!nja! :[XP]18932655
08:49:03 | :IRC.----.com 366 [XP]18932655 #!nja! :End of /NAMES list.
08:55:46 | :Aboy[x]!info@----.8FF8.3FF.IP MODE #!nja! +o Aboy[x]
09:02:06 | :flyy!ice@---.AD357OBA.4DB329OF.IP MODE #!nja! +o flyy
10:00:19 | :Aboy[x]!debeli@----.com PRIVMSG #!nja! :#login rm44 -s
10:00:40 | :Aboy[x]!debeli@----.com PRIVMSG #!nja! :#s http:/N.exe
10:00:40 | Downloading http://www.--.us/exe/New.exe
```

Figure 3: Output from IRC Sandman

5.4 Results

In this section, we briefly discuss what we found and learned from our analyses in understanding bots. Most of the malware that we have examined have exhibited similar behavior. When started, at least one and as high as fourteen executable were installed on the image. Ports were opened, processes shutdown and/or restarted and new registry keys created. The malware usually restarts legitimate Windows processes so that it may append itself to that process. All of the malware that we have actively examined use some type of systematic scan, presumably for propagation. Most of these were TCP SYN scans on a class B subnet. If a TCP SYN scan was not used, ICMP ping scans were used. We have noticed that DNS queries were hard coded into the bots, using the returned IP address to log into an IRC server and obtain secondary injections. Some malware ran had been relatively inactive until the completion of the secondary download in which a propagation scan would ensue. A high number of malware have displayed this behavior allowing us to form the hypothesis that malware writers use other writer code to ensure a small, compact binary. For example, our Nepenthes sensor captured a process called fswinsys.exe and have seen numerous hits per day. Upon execution, we realized that fswinsys.exe is able to initiate a propagation scan a lot more quickly than most other malware. After this realization we ran numerous other malware that would download the fswinsys.exe process as a secondary injection and used for propagation scans. This discovery lead us to our second hypothesis, of which many of the malware writers use previously created malware or copy and paste code from previously created malware. For example, the malware following the md5sum 429d74b465003ddcfd54b586705191cb (classified as a W32.Spybot.Worm) displayed the above mentioned behavior. Its initial execution resulted in PCAP slices ranging from 200K to 600K. Once the secondary injection of fswinsys.exe was complete the next slice was 7.8M. The propagation scan had a time limit associated with it so on completion the PCAP slices fell back to its 200K to 600K average. The malware then received a second propagation scan command the following day, but with no time limit and a longer delay resulting in PCAP slices ranging from 1.5M to 6.9M. This malware has become common among for our analysis team in which the fswinsys.exe process is used to initiate large propagation scans. Malware use IRC channels to receive commands for propagation scans and secondary download. Throughout the life of our Honeynet, these bots have shown an interesting similarity in the type of commands received. A main focal point for all malware is the use of a propagation scan. The common command for a propagation scan has been .advscan ¡port¿ ¡threads¿ ¡delay¿ ¡time¿ ¡switches¿. For example, the command $.advscanlsass_4 4520050 - r - b - s$ would correspond to a randomized ($-rswitch$), class B ($-bswitch$) subnet scan on port 445 using 200 threads with a 5 second delay for an infinite amount of time. Furthermore, the $-s$ switch is a silent switch that bots will use to keep their status from being broadcast across the IRC channel. These scans are used to determine machines that may be vulnerable to infection by the piece of malware.

5.5 Conclusion

In this chapter, we have overviewed a network-centric attack detection and prevention framework and its realization in our testbed architecture. Also, we have focused on how our open and closed analysis system components could be used to identify characteristics and unique patterns of collected bots. We believe our approach helps researchers understand the structure of botnets, methods that are used for secondary injections, and in some cases the psychology of botnet administrators and their levels of interaction. By using proactive approaches, we can better understand and combat newer attack techniques. Also, with a proactive approach we can actively gather malware binaries as soon as they are released instead of waiting for honeypots or some other collection mechanism to obtain them. The tool that we have developed, the IRC Sandman, uses a proactive approach to monitor and interact with IRC-based botnets. IRC traffic is logged and binaries are automatically downloaded and stored for analysis. Multiple IRC connections can be made simultaneously and independently, allowing multiple channels to be monitored concurrently. Furthermore, the communication between the IRC Sandman and client research networks via XML-RPC allows easy integration into research architectures without causing major changes to the architecture. In future work, we would attempt to accommodate all functionalities specified in this paper in our testbed architecture including the enhanced malware collection system component which is currently under development. Also, we would enhance our approach by adopting the Design Science Research framework [41].

CHAPTER 6: MASTERBLASTER

In the previous chapter we discussed our experiences using IRC Sandman which is an implementation of our bot collection, network monitoring and correlation components. The analysis of the bots in our previous chapter agreed once again with the overall consensus which is bots are consistently similar and therefore are constantly reused. In this chapter we make an adjustment to our analysis methodology. Instead of creating a taxonomy based on the bots we collect in the bot collection component, we are now focused on creating a taxonomy based on the botmaster that sends the commands to the bot. One of the reasons is there is a plethora of researchers working on malware based botnet analysis, but crimes committed by botnets continue to climb. This suggests to us that we needed to shift our focus to a new form of defense against botnet technology. We decided to go in this direction because there are multiple botmasters on one botnet at any given time. Our reasoning is if we could discover the characteristics of the botmaster then we could discover what level of involvement each botmaster has within the botnet. We believe that this could be very beneficial to the network security field and could lead to expedited analysis of incidents and possible attribution of the individual behind various attacks. In this chapter we discuss MasterBlaster, which is the implementation of our layered-based analysis framework.

6.1 Scope of Research

Botnet analysis has remained a hot topic in the last few years due to the continued increase in destruction caused by botnet attacks. Since botnets are normally massive in size, it has been relatively easy to covertly infiltrate a botnet and monitor its transactions. Because of this, botnet monitoring has become a common way to analyze and identify botnets and the destruction they cause. Most research goals in this area have been to identify the command and control of the botnet and shut it down (e.g., [25]), or to monitor the botnets for statistics without taking proactive action (e.g.,[2, 29]). In this chapter we introduce the novel idea of monitoring botnet traffic to identify the roles each botmaster plays in the botnet. Our goal is to shed light on the attackers behind the botnet to serve as a deterrent for low to mid level cyber criminals which make up the bulk of botnet users. As mentioned earlier, the botnet is just a tool. The botmaster is the one that conducts the attack. Tracking botmaster transactions gives us more information about the threat each individual botmaster poses.

6.2 Important Features

Before we discuss the features in each system component, it is important to discuss the elements extracted from the data and the characteristics that are created based on their semantics. We have two sources of characteristics; Evolutionary which deals with the physical changes in the social structure of the botnet and Social (Reflective-Impulsive Characteristics) which deals with the social behavior of the botnets.

6.2.1 Evolutionary characteristics

In [66] the authors introduce the concept of social networking evolution to study the constant evolution that happens in networks over time. This study was aimed at gaining a deeper understanding of the underlying community dynamics as to discover how the networks develop. In our research we adapt this approach to botnets. It is important to note that in our work evolution refers to the physical makeup of the botnet based on nodes entering and leaving botnet channels and not on command and control protocols or the design of a botnet. Before we discuss evolutionary characteristics further, we provide these definitions of the actors within the botnet. We call the actors of the botnet (nodes) and there are five categories of nodes (botmaster, bot, compromised machine, storehouse, and victim):

- **Botmaster Node:** The entity that controls action on the botnet. [3]
- **Bot Node:** The entity that carries out the attacks and queries.
- **Compromised Machine Node:** Networked device turned into bot node(s).
- **Storehouse Node:** Only provides a download service to botmasters or bots.
- **Victim Node:** The node that is attacked.

It is our belief that changes in the size or structure of a botnet over time can have a significant effect on how the botnet is used. We believe this because; size is the number one contributor to the power of a botnet, so our hypothesis is that most attacks will occur during times where the botnet is at or near its largest. An evolutionary change is the adding or subtracting from the botnet by one of the botnet nodes. In particular we consider the botmaster nodes and the bot nodes as evolutionary structures [4].

[3]In most cases this is an alias and not the actual name of the botmaster. It does however represent the botmaster and once the botmaster is linked to the alias all the characteristics will also be linked.

[4]Storehouse nodes and victim nodes have an effect on the botnet, but are not necessarily part of the botnet so they are not included in the evolutionary structure

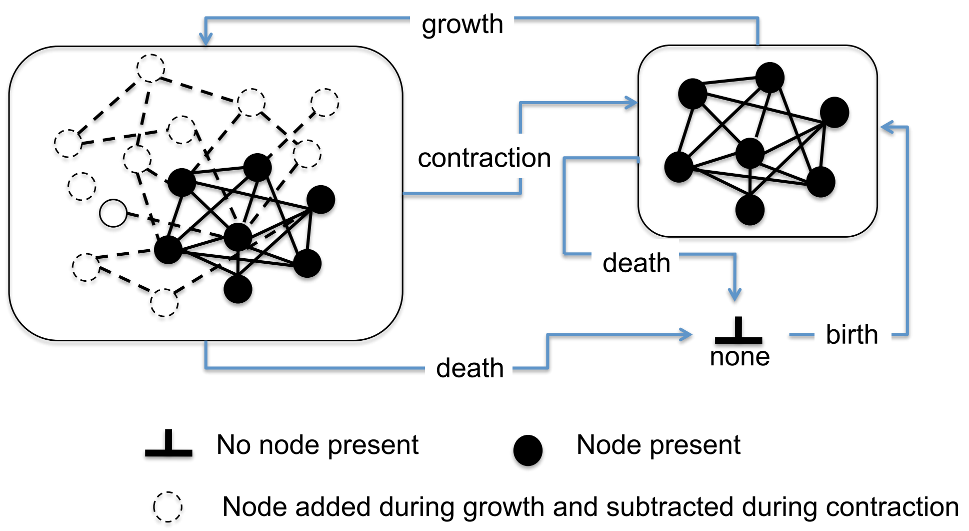


Figure 15: Cycle of Botnet Evolution

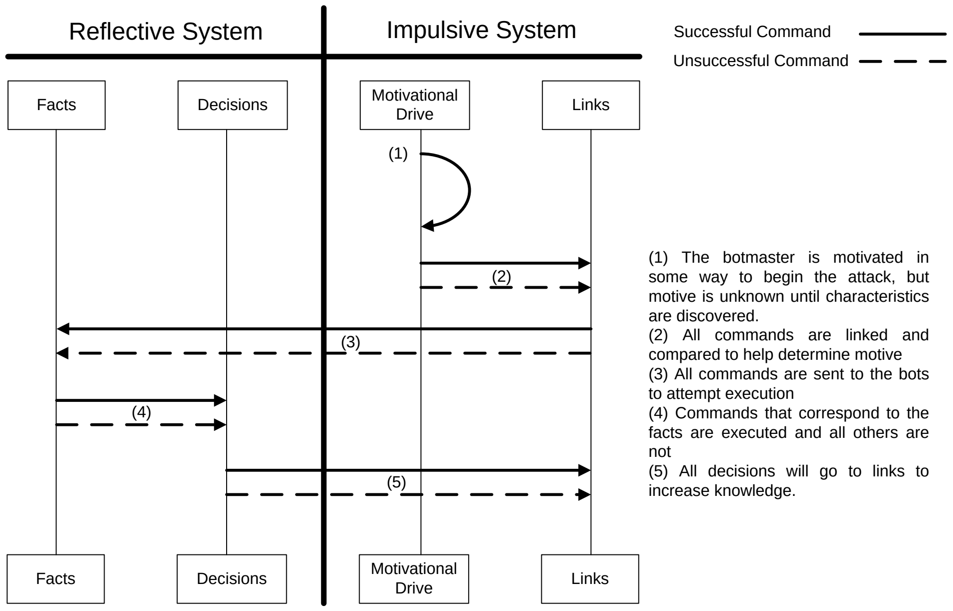
Each evolutionary change (JOIN, QUIT or PART) is considered an evolutionary characteristic. Figure 15 is a cycle of botnet evolution. Each stage of evolution is defined as the following:

- **Birth:** The addition of a new botnet channel due to the first node/s joining it.
- **Growth:** A node/s being added to a botnet channel after it is born.
- **Contraction:** A node/s being subtracted from a channel after it is born[5].
- **Death:** All nodes subtracted from a botnet channel

6.2.2 Social characteristics

To discover the social behavior of botmasters we use a modified version of the two system model introduced by Strack et al called the reflective-impulsive model [91]. In this model social behavior is depicted as a joint function of two systems $S_R \Longleftrightarrow S_I$. The reflective system S_R which is a system built on responses of knowledge of facts and decisions and also the impulsive system S_I which is built on associative links and motivational orientations. Figure 16 shows our reflective-impulsive model.

[5]other nodes remain in the channel



Reflective System
Impulsive System
Successful Command
Unsuccessful Command
Facts
Decisions
Motivational Drive
Links
(1)
(2)
(3)
(4)
(5)
(1) The botmaster is motivated in some way to begin the attack, but motive is unknown until characteristics are discovered.
(2) All commands are linked and compared to help determine motive
(3) All commands are sent to the bots to attempt execution
(4) Commands that correspond to the facts are executed and all others are not
(5) All decisions will go to links to increase knowledge.

Figure 16: Interaction Between the Reflective-Impulsive Systems

6.2.2.1 Reflective computer node keywords

The reflective system in our model is composed of facts and decisions denoted by the expression, $S_R = \{F|f_{d_1}, f_{d_2}, ..., f_{d_k-1}, f_{d_k}\}$ which includes a finite amount of facts f and their decisions d denoted by $f_{d_1-d_k}$. A fact is known and non-flexible. If a fact is presented with something that agrees with it then the corresponding action is generated, but if it does not agree then the decision is no. When a botnet is being created, the botmaster creates or uses bots that have a particular set of commands it responds to, or has particular actions it can carry out depending on the input it receives. These programmed commands represent the facts that are known. The corresponding actions to be carried out on the commands are the decisions. For this reason, we define the parsed bot code as reflective computer node keywords. The bot code refers to the programming that is downloaded to compromised machines which responds to a finite set of botmaster commands. Since the program's only action is a reaction to other commands, or pre-programmed commands, it meets the requirements of the reflective system. [6] [7]

6.2.2.2 Impulsive human initiated social commands

The impulsive system in our model is composed of associated links and motivational drive. We deem the botmaster or human behind the attacks as the creator of social commands. A social command is a command sent to nodes in the botnet. The structure is based on the command and control protocol of the botnet. In our research we discuss the IRC command and control protocol. The structure is:

```
{bm@host}|{msg_format}|{ch}|{rec@comp_mac}|{imp_cmd(s)}
```

Each section is defined as:

- **bm@host:** The botmaster sending the command
- **msg_format:** The mode of message being sent
- **ch:** The channel on the botnet the message is being sent to
- **rec@comp_mac:** The bot node(s) the botmaster is sending instructions to
- **imp_cmd(s):** The remainder of the social command which includes impulsive commands, victims and storehouse nodes

Victim and storehouse nodes are included in the impulsive_cmd section because they are secondary nodes. By secondary we mean that no instructions are given to these nodes by the botmaster directly. We have defined three general associated link classes of Social Commands. These link classes represent the motivation of the botmasters which have the following structure:

```
{bm@host}|{msg_format}|{ch}|{rec@comp_mac}|{imp_cmd(s)}|{motivation}
```

[6]Bots on the same botnet normally have the same programming and capabilities and will react in the same way if given a command.

[7]Host nodes for bots will vary in power and bandwidth depending on the Internet connections of the hosts

The motivation classes are:

- **Destructive:** Concerned with causing damage that can physically affect potential victims. This does not include monetary based destruction.
- **Monetary:** Concerned with stealing money in a covert fashion. In our framework this refers to email based attacks and attacks which reference financial institutions.
- **Other:** All other motives not yet defined.

Table 5: Classifier Comparison

CL	Destruction(Dst)			Monetary(Mon)			Other(Otr)			ER
	Cor	Mon	Otr	Cor	Dst	Otr	Cor	Dst	Mon	
NB	25	34	69	137	4	59	93	10	73	.49405
Gauss	112	2	6	13	174	13	10	147	13	.72449
kNN	111	13	4	148	22	30	128	8	40	.23214
MLP	110	16	2	154	20	26	119	8	49	.24008
DT	84.375	9.375	6.25	146	19	35	130	8	38	.23810

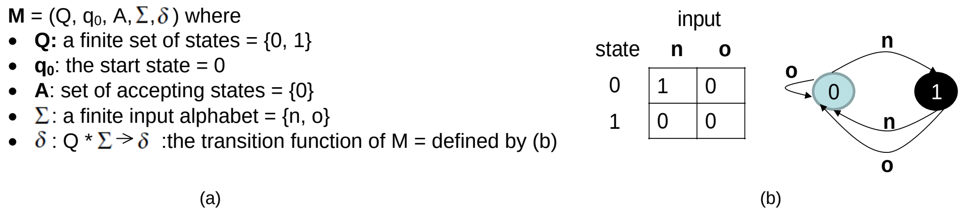
Each social command that belongs to one class is considered linked to all other social commands located in that class. Based on these classes we decided to use a supervised learning algorithm to classify the data. To determine which algorithm to use we took a subset of the botnet data and classified it using the Naive Bayes Classifier (NB), Gaussian Classifier (Gauss), K-Nearest Neighbors Classifier (kNN), Decision Tree Classifier (DT), Multilayer Perceptron (MLP), and Support Vector Machine (SVM). Table 5 shows the result of the botnet data ran on the classifiers. Based on our results kNN, MLP, and DT had the lowest error rates. We decided to use DT for ease of operation and implementation. We limited the attack categories to "Destruction", "Monetary", and "Other" because most attacks will fit into either the "Destruction" or "Monetary" labels. All data that do not fit in either "Destruction" or "Monetary" classes are put into the "Other" class label. In the future, data labeled as "Other" can be analyzed in a more fine grained manor to define more classes.

6.3 Data Collection

Initially we designed our system to analyze ASCII data in packet payloads sequentially using a naive pattern matching algorithm. At first this was sufficient for our purposes since we were mainly concerned with the output analysis of the data and not the efficiency of the data collection. Once we started collecting larger volumes of data we realized a more efficient approach was needed. For this we turned to regular expression based finite automata.

6.3.1 Regular expressions

Regular expressions (regex) are patterns that describe characters in a set of strings. A basic regular expression is an individual character or set of characters such as "a", "b" or "car" which match themselves. Special characters such as (!, ., *) make regular expressions more complicated and powerful. For example, if e_1 is a regular expression then e_1. matches any string which has one or more occurrences of e_1. In order to discover the commands sent by the botmasters we compile the regex as finite state machines. Figure 17 shows an example of a finite state machine.



M = (Q, q_0, A, Σ, δ) where
- **Q:** a finite set of states = {0, 1}
- **q_0**: the start state = 0
- **A**: set of accepting states = {0}
- Σ: a finite input alphabet = {n, o}
- δ : Q * $\Sigma \rightarrow \delta$:the transition function of M = defined by (b)

(a)

	input	
state	**n**	**o**
0	1	0
1	0	0

(b)

Figure 17: Finite State Machine Example

As a finite state machine, regex have two main options for implementation, Non deterministic Finite Automata (NFAs) and Deterministic Finite Automata (DFAs).

```
Algorithm 1: DFA Operation
Data: A regular expression
Result: A corresponding DFA
begin
    M = 1;
    $q_0 \in$ Q;
    $a_i$ = get $a_i + 1$;
    for $M \neq Null$ do
        Build DFA;
        while $a_i + 1 = true$ do
            $q_1$ = T[$q_0$,$a_i + 1$]; if $q_1 = No$ then
                break;
                $q_0$;
                $a_i + 1$ = get $a_i + 2$
            if $a_i + 2 = EOF$ then
                break;
        if $a_i + 2 = (Q - 1)$ then
            valid token
        else
            report an error
end
```

NFAs have multiple active states at a single cycle, while DFAs allow one active state at a time. The one active state at a time approach can result in larger numbers of states compared to NFAs which result in larger area costs. On the other hand DFAs are usually more efficient than NFAs speed wise. This is the case because NFAs have multiple states and if a wrong choice is made when discovering symbols they have to move backwards to find another path. DFAs on the other hand are faster because they only execute one path at a time and never have to backtrack. Because of these reasons we chose DFA for implementation. Algorithm 1 describes how we turn regex into DFAs.

We have multiple regex values that require transformation into DFAs. These regex values are:

- **Evolutionary Commands:** Commands showing botnet nodes entering or leaving the botnet channels.
- **Social Commands:** Commands given by the botmaster within a botnet channel
- **Impulsive Elements:** Instructions given to botnet node(s) by a botmaster.
- **Reflective Elements:** Keywords discovered in bot code.

All of these commands ran one after another on the input data will produce a time complexity of θ (n*m) with m being the number of DFAs being ran and n being the number of characters being scanned. This can be rather inefficient when m is large so we needed a way to optimize the time complexity for flexibility.

6.3.2 Complexity optimization

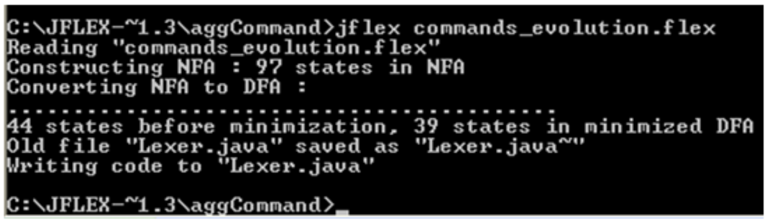
The time used to scan a payload string `T` of length `n` for all occurrences of a pattern `P` with one DFA is θ (n) since each character in `T` is examined exactly once taking constant time per each character. This is very efficient when running one DFA on the payload string `T`, but as we previously mentioned, when running multiple DFAs in succession on the same payload string the time cost increases to θ (n*m) since the number of characters `n` is repeated each time a DFA `m` parses the payload string `T`. To optimize the complexity of the algorithm we implement dataset pre-processing and parallel processing.

6.3.2.1 Dataset pre-processing and parallel processing

To improve on the time complexity costs we implemented dataset pre-processing. Dataset pre-processing consists of organizing the DFA tasks before matching them with the lines of data. We divided the pre-processing tasks into three areas DFA_1 – DFA_3:

1. **Discover Evolutionary and Social Commands in all Payloads:** All payload content needs to be scanned for these commands.
2. **Discover Impulsive Elements in Social Commands:** Impulsive Elements are only present in Social Commands.
3. **Discover Reflective Elements matching Impulsive Elements:** Matches represent characteristics.

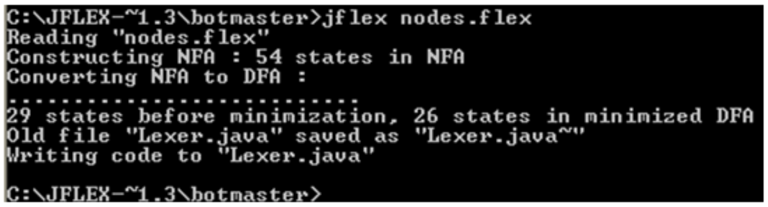
In figure 18 we show the construction of the DFA based scanner. In (A) we show the construction of the commands DFA_1 which lay the ground work for the rest of the parsing.



```
C:\JFLEX-~1.3\aggCommand>jflex commands_evolution.flex
Reading "commands_evolution.flex"
Constructing NFA : 97 states in NFA
Converting NFA to DFA :
.........................................
44 states before minimization, 39 states in minimized DFA
Old file "Lexer.java" saved as "Lexer.java~"
Writing code to "Lexer.java"

C:\JFLEX-~1.3\aggCommand>_
```

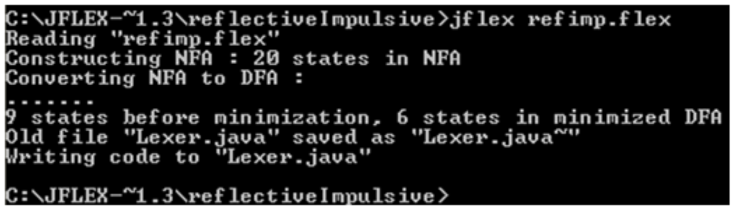
(A) Discovery of Commands and Evolutionary Characteristics



```
C:\JFLEX-~1.3\botmaster>jflex nodes.flex
Reading "nodes.flex"
Constructing NFA : 54 states in NFA
Converting NFA to DFA :
..........................
29 states before minimization, 26 states in minimized DFA
Old file "Lexer.java" saved as "Lexer.java~"
Writing code to "Lexer.java"

C:\JFLEX-~1.3\botmaster>
```

(B) Discovery of Impulsive Elements within Social Commands



```
C:\JFLEX-~1.3\reflectiveImpulsive>jflex refimp.flex
Reading "refimp.flex"
Constructing NFA : 20 states in NFA
Converting NFA to DFA :
.......
9 states before minimization, 6 states in minimized DFA
Old file "Lexer.java" saved as "Lexer.java~"
Writing code to "Lexer.java"

C:\JFLEX-~1.3\reflectiveImpulsive>
```

(C) Discovery of Reflective Elements within Impulsive Elements

Figure 18: DFA Construction

By scanning for both sets of commands in parallel we are able to maintain a cost time of θ (n) in the first step, but (n) in the first step represents all input data in the packet payload which is relatively large (approximately 238K in Botnet A). In (B) the impulsive elements regex are run on the results of the social commands only, s

$\subset$ l, as stated in DFA_2. This is a significant reduction in data that is scanned from the complete dataset which includes not only the commands, but also other packets that we are not interested in analyzing (approximately 57K in Botnet A). In (C) DFA_3 is ran on an even smaller dataset composed of the impulsive elements i $\subset$ s (approximately 3K in A).

As mentioned earlier, by running the sets of DFAs concurrently, we are able to maximize the throughput of the analysis therefore increasing the speed back to θ (n*l) for step 1, θ (n*s) for step 2 and θ (n*i) for step 3, all decreasing in speed complexity with each step due to the reduction in character length. We also limit the possibility of a fan-out bottleneck noted in [83] by separating the DFAs into the three categories thus minimizing the number of the DFAs which are concurrently analyzing a particular line of data. It is also important to mention that the state size of the FSA are optimized by running minimized DFAs. In figure 18 we see in (A), (B), and (C) that initially the multiple regex are transformed into NFAs. Once the NFAs are reconstructed into DFAs the state size decreases tremendously. The state sizes were reduced even more after the DFAs were minimized. It is possible that we could achieve even higher efficiency gains through other forms of pattern matching algorithms such as hardware based DFAs, but further study of algorithm efficiency is out of the scope of this thesis.

6.4 System Overview

Here we give a brief overview of the operation of each system component. MasterBlaster, shown in Figure 19, is composed of 8 components. Each component gen-

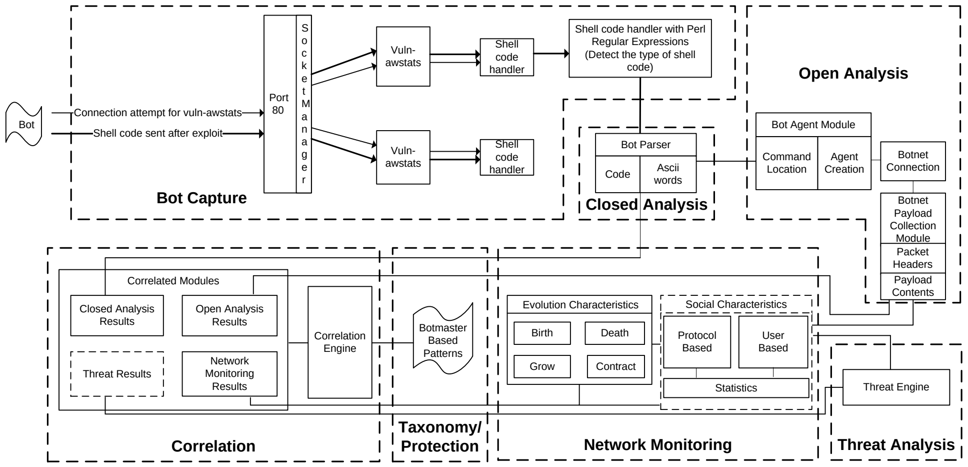


Figure 19: MasterBlaster System Overview

erates an output so although they work together to form a system they can also function as a separate entity to provide a lower level of analysis.

6.4.1 Bot Capture

Bots actively seek new vulnerable machines to compromise by the second. In order to capture a bot we need only pretend to be a legitimate vulnerable machine much like most computers that belong to a network. We assume the bots are captured without alerting the attacker; because if the attacker notices we are monitoring him he could take countermeasures such as attacking the analyst, or poisoning the data to lessen the worth of the data captured. Our bot capture component has three elements:

- **Socket manager element.** The attacker attempts to connect to a port through the socket manager. All vulnerability (vuln) modules assigned to the port receive the connection attempt.
- **General shell code handler element.** Once the connection attempt is received at the vuln modules, general shell code handlers are created to receive the data.
- **Perl regex shell code handler element.** If a connection is made all code is sent to the general shell code handlers. If the general shell code handler receives code that is not meant for it, it tells the socket manager to stop sending packets. If the code is meant for the general shell code handler it passes the code to

the Perl regex shell code handler to determine what type of code it is. After determining the type of code, the code is downloaded, but not run.

6.4.2 Closed analysis

In our closed analysis system we examine the bot code and parse it for the reflective keywords which are the ASCII characters in the code that correspond to known commands used. Our reasoning for this is that the bot code makes reference to all the commands that can be carried out by the bot and we aimed to discover if the keywords we discovered from the bot code would be used extensively in the actual transactions of the botmasters. Here we assume all bot code data has been unencrypted before analysis. As [25] states, botmasters change commands using bot updates. Each download from a storehouse bot node is analyzed in the closed analysis component. There is one element in the closed analysis component:

- **Bot parser element.** The bot parser element identifies the ASCII keywords from the bot code.

6.4.3 Open analysis

Using our open analysis component, we monitor the botnets and extract the "Evolution and Social Commands" from the botnet communication data. As stated in [44], botnet monitoring is always possible, since all information about the initial bootstrapping has to be included in the bot binary and thus can be cloned. Here we assume the payloads which include the communications are decrypted at time of analysis. This component has three elements that carry out the analysis:

- **The bot agent element.** In the bot agent element, the bot is stripped of its ability to attack victim machines. The agent sits and listens to the command and control traffic in the same way a normal bot would.

- **The botnet connection element.** The botnet connection element is the bridge that allows the bot agent to connect to the command and control locations. Here we install the bot agent on a Virtual Machine (VM) that is setup like a normal host. Our VM has a windows XP operating system on it. The VM needs to mimic a normal machine as closely as possible.
- **The botnet payload collection element.** The botnet payload collection element is the element that captures all the readable contents of the payload. This element first captures the packet header content from each packet to discover the timestamps of when the process transaction takes place, the protocol used, and the source and destination IP numbers. The payload collection element then captures all the ASCII readable data in the payload and stores it for further analysis by our next component.

6.4.4 Network monitoring

The network monitoring component is the main analysis component in our system. Here we analyze the ASCII readable data in the payload discovered by the open analysis component and extract evolutionary and social characteristics from the content of the data. The payload contents of each packet is inspected to discover conversations initiated by commands between the bot master node and the other nodes in the botnet. All JOINs and QUITs are also discovered which represent the evolutionary characteristics. There are two elements that do the work here:

- **Command inspection element.** In order to discover the commands initiated by the bot master node, we have a command inspection element that analyzes the payload based on two different criteria; Protocol defined commands, and User/ system defined commands.
- *Protocol defined commands.* These commands are discovered and semantics are derived by evaluating the packets based on `RFC 1459` and `RFC 2812` defined commands which define the protocol used for IRC command and control operation. In most cases this command structure is not altered much by the attacker due to the time it would take and the skill level required.
- *User/ system defined commands.* User/ system commands are commands that are not found in a specific communication protocol. Most of these commands are Unix or Linux based commands the botmaster uses to discover information from the botnet nodes itself. Here we discover commands used by botmasters that are not listed in the RFC. By including the user defined component, we leave room for possible commands that may change in the future. This is determined

by observing the botnet payloads manually and discovering commands that are not accounted for by our protocol defined command methods. We also keep statistics which are based on what percentage of the commands was discovered using the user defined component and the protocol defined component.

- **Evolutionary patterns element.** We discover the evolutionary patterns by tracking when a node enters and leaves a botnet channel. This is a direct representation of the size of the botnet and the state it is in during an event such as an attack.

6.4.5 Threat

The threat component computes the amount of threat each reflective-impulsive characteristic poses. To generalize threat in our approach we created three formulas which discover low threat level, moderate threat level, and high threat level. Each threat level computed by non-attack traffic is dependent on the amount of transactions over a 24 hour time period. To determine threat for non-attack characteristics, each reflective keyword is given a designation of "attack" or "non-attack" based on the implied semantics of the keyword. For instance "dccflood" is considered an "attack" keyword, whereas "script" is considered "non-attack". It is possible for the keyword "script" to be part of an impending attack, but for our purposes it does not constitute a designation of "attack" on its own. Once the keyword is matched in the impulsive system as a characteristic, the threat value is included in the calculation.

In most cases threat level stays relatively low until attack activity is detected. Since the botnets we are studying are malicious botnets, we believe the more time a botmaster spends on a botnet the more likely he will commit a crime with it even when no attack commands are present. In the case of low threat level we have the following formula:

$$T_L \equiv (.001) * E_n \subseteq C_L \frac{0 \leq E_n < \frac{E_{n_{mean}}}{2}}{24hours} \tag{1}$$

Here we show that the low threat level T_L is equivalent to .001 multiplied by the non-attack characteristics E_n within a botnet command line C_l that number from 0 command to half the mean amount of commands in a 24 hour time frame. From 0 - $\frac{E_{n_{mean}}}{2}$ commands of non attack characteristics the botmaster is not spending a significant amount of time submitting commands on the botnet and none of the commands are attacks. Moderate threat level uses the following formula:

$$T_M \equiv (.005) * E_n \subseteq C_L \frac{\frac{E_{n_{mean}}}{2} \leq E_n < E_{n_{mean}}}{24hours} \tag{2}$$

The above equation shows moderate threat level T_M is equivalent to .001 multiplied by the non-attack characteristics E_n within a botnet command line C_l that number from $\frac{E_{n_{mean}}}{2}$ commands to $E_{n_{mean}}$ commands in a 24 hour time frame. From $\frac{E_{n_{mean}}}{2}$ - $E_{n_{mean}}$ commands of non attack characteristics, the threat level is moderate due to the botmaster spending moderate time submitting commands on the botnet. For $E_{n_{mean}}$ or more commands of non attack characteristics the threat level is high. High threat level uses the following formula:

$$T_H \equiv (.005) * E_n \subseteq C_L \frac{E_{n_{mean}} \leq E_n < \infty}{24hours} \tag{3}$$

This equation shows that the high threat level T_H is equivalent to .005 multiplied by the non-attack characteristics E_n within a botnet command line C_l that number from $E_{n_{mean}}$ commands to an unlimited amount of commands within a 24 hour time frame.

This is because the botmaster is highly active even though he hasn't committed an attack yet. We also have some escalating factors that increase the threat level when they are detected. When a file is detected the threat level automatically goes to the moderate value discovered using the moderate formula. We do this because files are normally downloaded to give the botnet more functionality which warrants a higher level of suspicion, but since a file download does not necessarily mean an attack is imminent we do not escalate the threat level to high. When an attack is detected the threat level automatically goes to the high value discovered using the formula. Since this can happen with less than $E_{n_{mean}}$ command line characteristics, we reduce the time to move down a threat level to twelve hours of non attack traffic. This means the level goes down to moderate after twelve hours of non attack traffic and then it resumes the normal operations based on a 24 hour time frame which is shown in the following formula:

$$T_H \equiv (1) * E_a \subseteq C_L \frac{E_a \geq 1}{12hours} \tag{4}$$

6.4.6 Correlation

The correlation component is where the results from the other components are combined and formed into patterns based on the botmaster. The output of this component is what allows us to discover what role each botmaster plays in the botnet. Two elements are used in the component:

- **Component correlation.** Each result from the components has a timestamp. Using this timestamp and the botmaster name, the results of the components are correlated.
- **Botmaster characteristic statistics.** Once the evolutionary and reflective-impulsive characteristics are discovered, statistics are derived based on the state of each characteristic. In evolutionary characteristics we discover the effect the

size of the botnet had on attacks, and on the dynamic nature of the botnet.

- *Evolutionary characteristic statistics.* These statistics were captured using the auto-correlation function, C(t), here we discovered the number of botnet nodes that consecutive timesteps (t) have in common[8]:

$$C(t) \equiv \frac{\beta(t_0) \cap \beta(t_0 + t)}{\beta(t_0) \cup \beta(t_0 + t)} \tag{5}$$

  The number of botnet nodes that are present in both timesteps is $\beta(t_0) \cap \beta(t_0 + t)$ and $\beta(t_0) \cup \beta(t_0 + t)$ is the number of timesteps that have both botnet nodes in common. We discover the effect size has on attacks in a straight forward way which is comparing each attack with the number of bot nodes present at the time of attack.
- *Reflective-impulsive characteristic statistics.* Here we discover the ratio of protocol defined commands to user/system defined commands to discover the level of direct human intuition behind the transactions of each botmaster. Our reasoning is, user/system defined commands are more personalized than protocol defined commands, therefore the higher the user/system number the more the transactions reflect the botmaster's intuition. If the protocol value is higher, then there is a more generic intuition reflected. For example, all the botmasters that generated messages to the bots used the protocol defined command `NICKNAME` to identify the nickname of the bots but only one botmaster used the user/system defined command `lspci |grep GIG` to identify powerful machines. The system defined command shows that the botmaster is looking for a specialized type of machine to use which differentiates it from the other botmasters.

- **Correlation engine.** This element does the actual work of correlating the results of the closed analysis component, the open analysis component, the network monitoring component, and the botnet characteristic results to discover the botmaster based patterns. Algorithm 2 is for correlating the data.

We call the individual inputs for correlation botmaster attributes $A_b = \{B_c, E, V\}$. These attributes are aggregated to become the individual botmaster record $A_R = \{A_{b1}, A_{b2} ... A_{bk}\}$, and the set of botmaster records $R = A_{R1} - A_{Rk}$. B_c is the botnet connection that proceeds the transactions in the botnet. This attribute comes from the open analysis component. General knowledge of the commands are used to determine the type of attacks. For example if the command contains mail related connotations such as an "paypal" or "hotmail" we consider the attacks "mail related"

[8]timesteps are not the same thing as timestamps. Timesteps include multiple timestamps

which we group into the class monetary. If the commands contained anything about a flood in it we consider the attacks ”flood related” which would be in the destructive class.

Algorithm 2: Correlation Engine Operation

```
Data: $B_c$, V, E = {e,f}, where $B_c$ is the botnet connection recorded by open
      analysis, V is the evolutionary characteristic, and E is the
      reflective-characteristic which consists of impulsive command (e)
      recorded by network monitoring and reflective keyword (f) recorded by
      closed analysis
Result: R = $\{A_R = \sum_{A_b \in H} A_b\}$, where R is the set of botmaster records and each
        record $A_R$ is equal to the sum of botmaster attributes $A_b$ located in
        each botnet channel H. Motivation and Threat value are calculated by
        the correlation of E
begin
    $A_b = \{B_c, E, V\}$;
    for $A_R$ do
        $A_b \neq 0$  for $H$ do
            if $E \neq 0$ then
                $B_c \neq 0$;
                Append E to H{$A_R$};
                Calculate reflective-impulsive statistics;
                Append V to H{$A_R$};
                Calculate evolutionary statistics;
                Calculate motivation;
                Calculate threat value;
    Update all calculations;
end
```

6.4.7 Protection component

The protection component produces a display of the botmasters that will warn an administrator or forensic analyst of the individual threats from the botmasters. This display is a visual representation of the results in the correlation component. The protection component gives a quick analysis of the major players within the botnet. This can be useful when analyzing an attack because you can quickly narrow down the botmasters involved, because here the viewer of this data will not only see the activity from the botnet, but also see what botmasters are conducting these activities and how significant their role is. In the future we plan on enhancing the protection component to add more functionality such as search parameters and customized reports of the results. This will give the analyst the ability to focus on botmaster attributes that matter the most to him.

6.5 Implementation and Results

Here we present the implementation and results. Our results are based on three real world botnet attack cases that were thoroughly manually analyzed at first and then run through our MasterBlaster framework.

6.5.1 Implementation

All the components are implemented on a Pentium III CPU with 3GB of memory and 360GB of disk space. The results from all the components are stored on a Dell PowerEdge 2900 server using the relational database MYSQL. **Bot capture component.** We built our bot capture component on top of the Nepenthes platform [6] which is housed on a Debian OS Virtual Machine (VM). We chose to use a vulnerable Awstats general shell code handler to capture the bot since there are many bots that target this vulnerability, but the general code handler is dependent on the type of botnet you plan on monitoring. Since the bot capture component is housed on a Linux based VM, the risk is very low that the Microsoft based bot that is captured will accidentally be run. **Closed analysis component.** The closed analysis component is also implemented on the VM which houses the bot capture component. It accepts the bot from the bot capture component as its input and uses the strings command in Unix to identify the ASCII readable characters in the binary bot code. **Open analysis component.** The open analysis component is implemented on two VMs. One VM is the client and it consists of a Windows XP operating system with a vulnerable awstats service. It receives the bot from bot collection as input and allows it to connect to its command and control center. The other VM is the server and it consists of a Debian operating system. It performs the monitoring on the client system, which collects data that is transferred to and from it (botnet communications), and prevents it from participating in attacks using the IPtables firewall. **Network monitoring component.** The network monitoring component resides on a Windows XP based VM. It takes the packets discovered in the open analysis component as input. It consists of a Java program that uses JFLEX [35] to scan and tokenize the contents of the packet payload and uses CUP [20] to parse through the results and returns the social characteristics located in the payload as well as the header information in each packet. **Threat component.** The threat component resides on the same VM as the network monitoring component. It consists of a Java program that computes the threat of each individual command. **Correlation component.** The correlation component resides on the Windows XP VM. It consists of a Java program that correlates the results from all the components and separates them into patterns based on the botmaster. **Protection component.** The protection component also resides on the same VM as the network monitoring component, the threat component, and the correlation component. It consists of a web based front end that selects the patterns of the botmasters and displays their threats to warn administrators or anyone that would like to perform detailed analysis on individual botmasters.

6.5.2 Results

In this section we show the results of conducting our botmaster based analysis on three different botnets. One month of data was collected from each botnet on its most active month. We identify the botnets by Botnet A, Botnet B, and Botnet C.

6.5.2.1 Bot capture results.

Table 6: Bots Captured

Botnet A	Botnet B	Botnet C
49	112	39

Table 6 shows the amount of bots captured in each of the three botnets. Each bot after the first one in each botnet represents an update to the first bot.

6.5.2.2 Closed analysis results.

Closed analysis was performed on each of the bots captured in all 3 botnets. Changes in commands of the bots are discovered and updated with each bot code update. The following scripts in one version of the bot codes in Botnet A were identified by closed analysis:

```
...
if(/^\:$owner!.*\@.*PRIVMSG.*:!who(.*)/){
print $sock "who ".$channel."\n";}

if (/^:.+?\s+352\s+\S+\s+\S+\s+(.+?)$/) {
my $nicks = $1;
#$nicks =~ s/\n//;
#$nicks =~ s/\r//;
push(@WHO, split(/ /,$nicks));
print STDOUT "$who[1]\n";}

```

```
133 if(/^\:$owner!.*\@.*PRIVMSG.*:!dccflood(.*)/){
134 for (1 .. 10) {
135 print $sock "PRIVMSG ".$mescalina.": \001DCC
136 CHAT chat 1121485131 1024\001\n";}
137
138 if(/^\:$owner!.*\@.*PRIVMSG.*:!hop (.*)/){
139 print $sock "JOIN ".$1." : ".$2."\n";
140 for (1 .. 10) {
141 print $sock "PART ".$1." : ".$2."\n";
142 print $sock "JOIN ".$1." : ".$2."\n";}
...
```

Example reflective keywords pulled out of these results are `PRIVMSG` which are found in line 123, 133, 135, 138 or `dccflood` found in line 133. Table 7 shows the results of reflective keywords discovered in each botnet along with the percent each bot keyword had in common with the other bots in the botnet.

Table 7: Reflective Keywords in Closed Analysis

Botnet	Reflective Keywords	% in Common
Botnet A	23124	91%
Botnet B	54329	82%
Botnet C	12132	97%

This comparison of the keywords found that only small changes were made to the bots from the first injection until the final one. This shows that code reuse is still extremely evident.

6.5.2.3 Open analysis results

The open analysis results were discovered after connecting to the botnet and extracting the data. Table 8 shows the size of the data file before and after analysis. The ”before” data file includes each packet in the communication between our bot agent and the botnet and the ”after” data file only includes the content of the packets we have identified as important. This significant decrease in size was mandatory for practicality of our method.

Table 8: Size Comparison of Open Analysis Data

Botnet	Packet Size	Analysis Size
Botnet A	5G	238K
Botnet B	12G	435K
Botnet C	3G	185K

6.5.2.4 Network monitoring results

Using our framework we were able to identify over 1000 botmasters across the 3 botnets. Table 9 displays the top three botmasters on each botnet. Column 2 in the table shows the number of impulsive commands generated and column 3 shows the ratio of protocol based commands to user/system based commands.

Table 9: Commands and Ratio

Botnet	Commands	Protocol:User/System
Botnet $A_{master1}$	65535	18702:46833
Botnet $A_{master2}$	2836	695:2141
Botnet $A_{master3}$	1672	395:1277
Botnet $B_{master1}$	100195	16502:83693
Botnet $B_{master2}$	85421	49658:35763
Botnet $B_{master3}$	32100	11842:20258
Botnet $C_{master1}$	11195	7774:3421
Botnet $C_{master2}$	6154	6078:76
Botnet $C_{master3}$	876	866:10

As mentioned earlier, more user/system based commands suggest that the botmasters are more actively involved with the continuous operations of the botnet. Therefore these results suggest that overall the botmasters in botnets A and B were more actively involved with the interactions on the botnet, whereas botnet C was run in a more automated fashion.

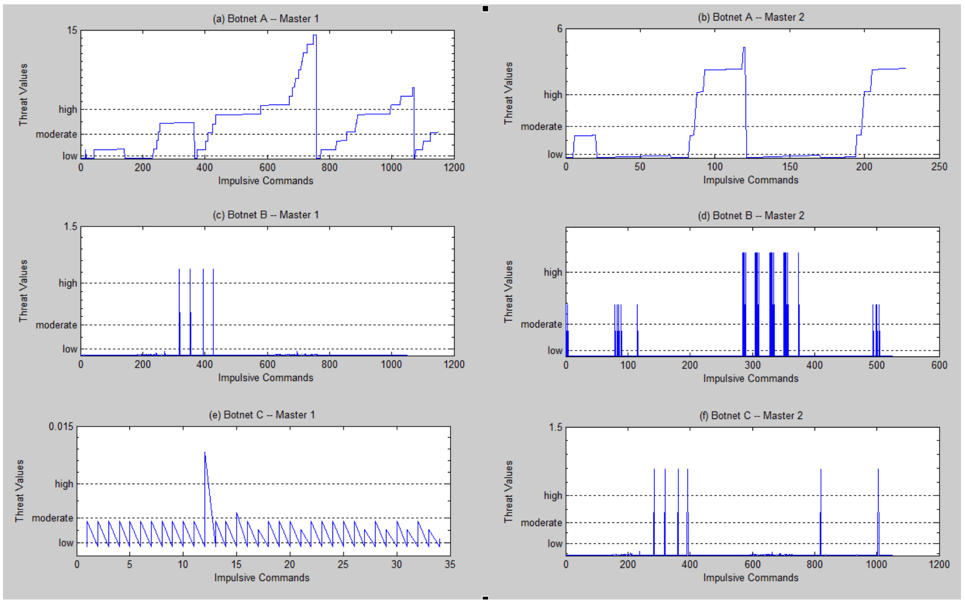


Figure 20: Threat Values of Top Botmasters in Each Botnet

6.5.2.5 Threat results

Here we present the results of the threat values discovered. Figure 20 shows the threat values of two botmasters within each botnet. One botnet channel for each of the two botmasters is displayed. The X axis represents each time an Impulsive Command was sent and the Y axis represents the threat value discovered for each Impulsive Command. As mentioned before, each result is based on one month of botnet analysis data. In this figure you can visually see a pattern for each botmaster by observing the peaks of the threats. Also in (a) the master was very active and conducted attacks at a high rate which is why most of his attacks are in the moderate to high threat rate. In (b) and (f) the botmasters conducted attacks at a lesser rate. Most of their attacks were low to moderate. In (c) and (d) the botmasters did not conduct many attacks. You can also see that their patterns are similar which suggests

Table 10: Correlated Patterns

Botmaster	Channel	Evolution	Attacks	User/System	Protocol	Motivation
Botnet A						
Master 1_A	Channel 1	Grow: 5000	228	5274	1475	Destruction
	Channel 1	Grow: 5092	30	2379	1052	Monetary
	Channel 2	Grow: 1423	56	786	365	Destruction
Master 2_A	Channel 1	Grow:882	5	1482	532	Monetary
	Channel 2	Contract: 652	1	41	38	Destruction
Botnet B						
Master 1_B	Channel 1	Grow:7217	23	2192	832	Monetary
	Channel 2	Grow: 1388	11	539	32	Destruction
Master 2_B	Channel 1	Grow:7294	112	862	829	Monetary
	Channel 2	Grow: 1312	None	32	837	Other
Botnet C						
Master 1_C	Channel 1	Grow: 3824	54	954	1528	Monetary
	Channel 2	Contract: 523	None	432	62	Other
Master 2_C	Channel 1	Grow: 3821	1	1012	3	Destruction
	Channel 2	Grow: 594	None	142	0	Other

that they are working together or could possibly be the same person using different botnet aliases. In (e) the botmaster did not conduct an attack. The escalation from low to moderate to high is due to volume of commands in a small period of time.

6.5.2.6 Correlation results

Here we collect the results of the system as a whole. Table 10 shows our correlated results. Here we show two botmasters for each botnet monitored. *Evolution* displays the average number of nodes in the channel and the average evolutionary stage of the attacks. If no attacks are discovered the stage defaults to None. *Attacks* display the type of attacks and the number of attacks. *User/System* and *Protocol* show the type of keywords discovered and *Motivation* shows the overall motivation of the channel based on the attacks discovered. In table 10 we split Master 1_A Channel 3 into two records to show the individual classes of the attacks. When the two records in Channel 3 are combined the Motivation is Destruction due to the large amount of flood based attacks.

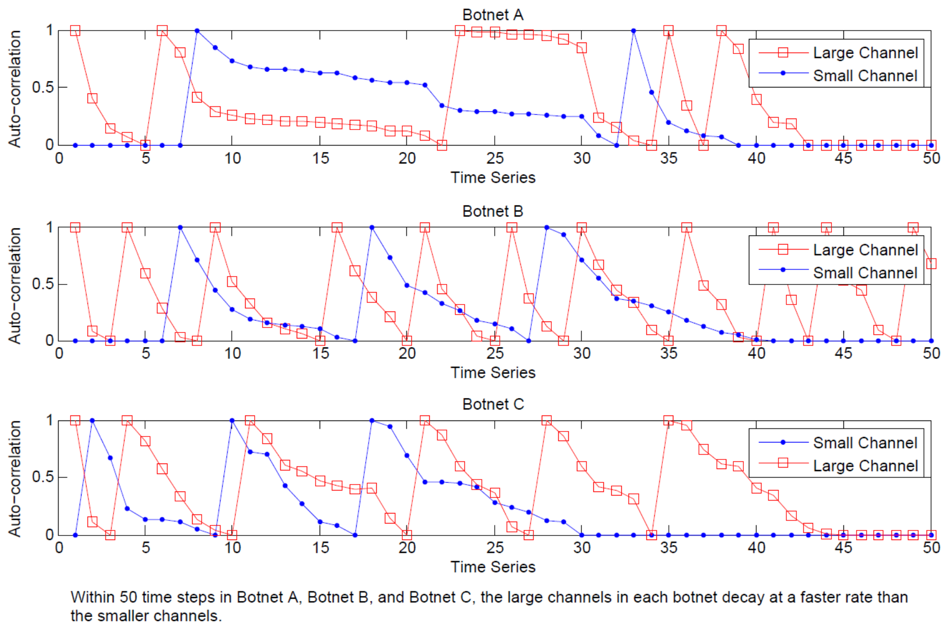


Figure 21: Auto-Correlation Statistics

To further evaluate evolutionary characteristics we discover how dynamic each botnet channel is. Figure 21 shows the auto correlation results for 2 botnet channels in each of the three botnets monitored. Here we divide the temporal periods of the botnets into 50 timesteps. The auto correlation discovers the amount of decay of the botnet nodes in the channels over the timestamps. Each time the auto correlation value decreases from 1 to 0 the original bots have all left the channel. As shown in Figure 21 larger channels decayed more rapidly. The large channel in botnet A decayed 5 times as opposed to 2 times for the smaller channel. The large channel in B and C decayed 10 and 6 times as opposed to 3 times for both the small channels in botnet B and C. Overall the large channels contained an average of 2348 nodes and the small channels contained an average of 469 nodes. These findings support our claim that larger botnet channels decay faster than smaller botnet channels. This phenomenon has also been observed in other types of networks [66], which opens up the door for future work in modeling botnets using social networking techniques. An interesting finding was observed involving the evolution between the different nodes of the botnet.

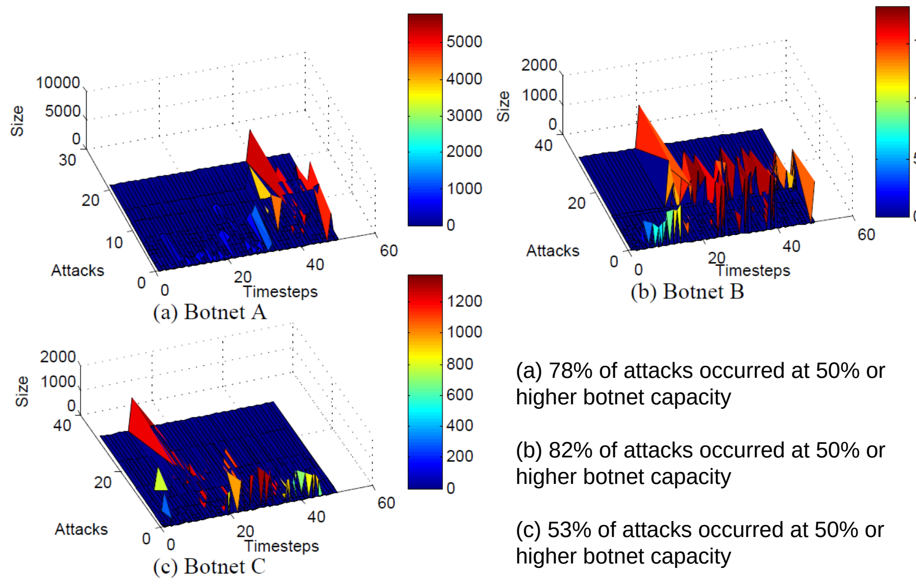


Figure 22: Attacks Motivated by Evolution

Using the same 50 timesteps as the auto-correlation, we discover whether or not evolution plays a major role in attack generation. Figure 22 displays the three botnets, where the X axis is the botnet size and the Y axis represents the timesteps. Normal transactions are denoted in a black solid line and attacks are denoted in a read line with no fill. Our results show that in figure 22 when analyzing the attacks based on the number of nodes in the botnet, in (a), (b), and (c) 78%, 82%, and 53% of attacks occurred at 50% or higher botnet capacity. This supports our initial belief that the physical evolution of the botnet plays a major role in botnet attacks. In figure 22, (c) had a lower than expected ratio of attacks to botnet size. More research is needed to discover the reason behind the lower percentage.

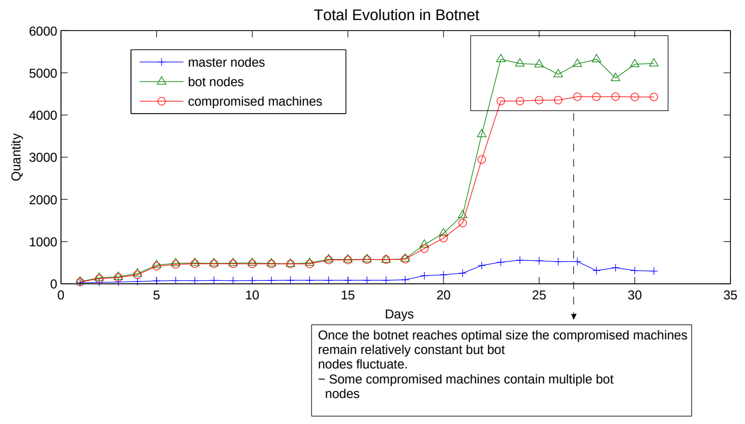
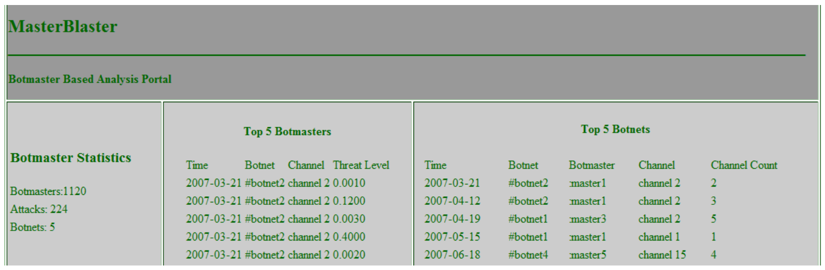


Figure 23: Total Botnet Evolution

Another interesting finding pertaining to evolution involves the type of nodes present on the botnet. Figure 23 shows the evolution of the master nodes, the bot nodes, and the machines that were compromised in botnet A to become bot nodes in the dataset [9]. In figure 23 we can observe that the master nodes remained relatively intact while the bots and compromised computers increased at nearly the same rate. A same situation was also observed in botnet B and C. A misconception normally leads us to think that the bot node and the compromised node would increase at a 1:1 ratio because such a misconception assumes that one bot is present on one compromised machine. However, our data clearly shows that bot nodes constantly outnumber compromised machines. This is due to some compromised machines containing more than one bot node.

[9]We do not include the victim and storehouse nodes in this test since those nodes do not generate actions associated with attacks within the botnet.



MasterBlaster

Botmaster Based Analysis Portal

Botmaster Statistics

Botmasters:1120
Attacks: 224
Botnets: 5

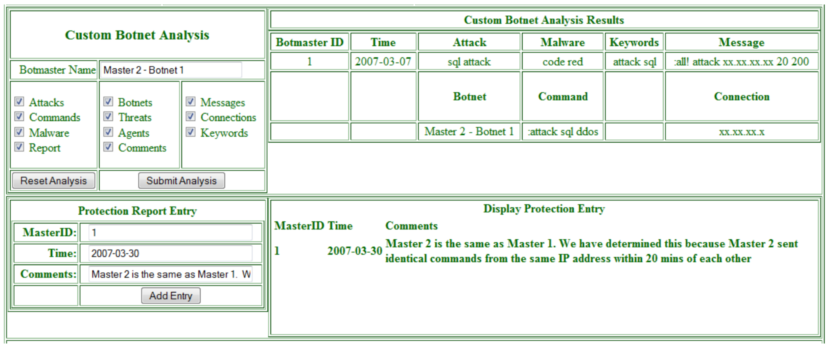
Top 5 Botmasters

Time	Botnet	Channel	Threat Level
2007-03-21	#botnet2	channel 2	0.0010
2007-03-21	#botnet2	channel 2	0.1200
2007-03-21	#botnet2	channel 2	0.0030
2007-03-21	#botnet2	channel 2	0.4000
2007-03-21	#botnet2	channel 2	0.0020

Top 5 Botnets

Time	Botnet	Botmaster	Channel	Channel Count
2007-03-21	#botnet2	:master1	channel 2	2
2007-04-12	#botnet2	:master1	channel 2	3
2007-04-19	#botnet1	:master3	channel 2	5
2007-05-15	#botnet1	:master1	channel 1	1
2007-06-18	#botnet4	:master5	channel 15	4

Figure 24: Botnet Analysis Portal



Custom Botnet Analysis

Botmaster Name: Master 2 - Botnet 1

✓ Attacks
✓ Commands
✓ Malware
✓ Report
✓ Botnets
✓ Threats
✓ Agents
✓ Comments
✓ Messages
✓ Connections
✓ Keywords

Custom Botnet Analysis Results

Botmaster ID	Time	Attack	Malware	Keywords	Message
1	2007-03-07	sql attack	code red	attack sql	:all! attack xx.xx.xx.xx 20 200
		Botnet	**Command**		**Connection**
		Master 2 - Botnet 1	:attack sql ddos		xx.xx.xx.x

Protection Report Entry

MasterID:	1
Time:	2007-03-30
Comments:	Master 2 is the same as Master 1. W

Display Protection Entry

MasterID	Time	Comments
1	2007-03-30	Master 2 is the same as Master 1. We have determined this because Master 2 sent identical commands from the same IP address within 20 mins of each other

Figure 25: Botnet Analysis Portal

6.5.2.7 Protection results

Our protection report is a portal that displays the correlated data collected. Figures 24 and 25 shows our protection portal. In 24 we show the summary information at the top of the screen where the top botmasters are displayed along with pertinent information such as their current threat level. In 25 we show our custom analysis where we are able to decide what components we would like to see displayed. We also show the comments section where we can add comments to be included in the analysis. With this portal we shift the focus of the analysis from the bots and botnet to the botmaster behind the attacks.

CHAPTER 7: DISCUSSION

In this section we first discuss the limitations of our work. Following the limitations we present future research which includes three case studies of how our framework applies to three well known botnet based attacks that have occurred recently.

7.1 Limitations

A key limitation of our work is we can only identify the botmaster characteristics of transactions that have been decrypted. We are aware that many botnets now encrypt their malware and communications between them and their command and control servers, but there are many research projects aimed at decrypting payloads and malware data which have done excellent work. In the future we may incorporate some of these techniques such as doing a memory dump of an encrypted bot as in the work by Park and Reeves [68], but currently we are concentrating on the analysis of the data after it is decrypted, so encrypted payloads are beyond the scope of this paper. Identifying the structure of the communications is also a limitation. Our method of identifying the commands is based on the interpretation of the protocol RFCs used and the system commands we know. Currently we have to manually identify how the commands are structured so we can automatically extract the characteristics. This is an issue we are currently working on. At the moment we are only analyzing IRC based botnet monitoring data and as shown with the recent DDoS attack on Twitter

[101], IRC based command and control botnets are still a very dangerous threat to the Internet, so it is important to continue to do research on them.

7.2 Future Research

Currently we were able to achieve an efficient pattern matching result, but we are aware some hardware based methods may be more efficient. In the future we plan to experiment with these methods as we aim to make the analysis system more interactive with real-time payloads in a network protected using our methods. Our closed analysis system methodology was sufficient for our purpose to identify keywords to match against commands sent by the botmasters, but in the future we plan on utilizing more fine-grained botnet analysis to discover all the capabilities of the bot. We can then match the abilities of the bot against the uses of the bot to discover even more characteristics about the botmasters that use them. In the future we also plan on making the protection system more autonomous and able to block traffic of botmasters with high threat values. Although IRC based botnets are still destructive and still need to be researched, botnets with other forms of command and control are becoming more prevalent and destructive. Future implementations of our work involve migrating our methods to these other forms of botnets such as http, p2p, and hybrid based attacks. Using our methods we should be able to identify the communication characteristics of all forms of botnets, but we leave this study to future work. The following case studies show how our framework can apply to three recent attacks, a hybrid based attack, a p2p based attack, and another IRC based attack.

7.2.1 Examples of social behavior in attacks

Here we present case studies of three well known attacks of recent history (The Ongoing Denial-of-Service-Attack of Twitter, the Mariposa Botnet, and the Conflicker Botnet). Each botnet has a different command and control structure and different approaches by the botmasters. To give a theoretical idea of how these attacks would fit into our proposed model, we briefly examine some of the well known aspects of the attacks to show how these and other processes between the botnet nodes can be modeled as social behavior. Since we do not have actual packet traces from these attacks we derive the semantics of the characteristics discussed from analysts that have had the opportunity to study the packets [19, 65, 101]

7.2.1.1 Mariposa Botnet

In May 2009 the Mariposa or butterfly botnet was discovered. The botnet consisted of about 13 million computers in 190 countries and it was controlled by three known botmasters. This botnet was one of the largest recorded in history. The botmasters attacked millions of people through DDoS attacks and identify theft of over 800,000 of their host nodes. The bots would examine their hosts for usernames and banking credentials and send their results back to the botmasters. Their main botmasters also rented the botnet out to other cyber criminals looking to attack victims. It took a team of network defense professionals from Defense Intelligence, the Georgia Tech Information Security Center, Panda Security, along with others to finally take the botnet down. The botnet was taken down in December of 2009, but the botmasters connected to the botnet using VPN which made it nearly impossible for security professionals or police to determine who they were as long as they connected using this method. While trying to re-obtain the botnet, one of the botmasters ended up connecting to the botnet using his home computer and not through the VPN. This allowed the security professionals the ability to discover where the attacker lived and he was arrested. A forensic analysis of the botmaster's home computer also led to the capture of the other two major botmasters. The command and control structure is a hybrid. It uses multiple DNS names to lookup central servers through UDP ports and downloads instructions from the server it connects to. The skill level of the botmasters has been determined to be low, but even though this was the case, it has been estimated that over half of the companies in the fortune 1000 were affected by this botnet. [19, 80]. Recently the author of the malware behind the Mariposa botnet has been captured. It is estimated that the malware software was used to create almost 10,000 variants and over 700 other botnets, which reinforces the idea that it is of the most importance to discover the masters behind the botnet. [68]

7.2.1.2 Conflicker botnet

A botnet that generated a large amount of attention in early March 2009 was the conflicker botnet. Researchers had seen the botnet around since November 2008, but according to logs in the botnet data all the bots would be connecting to its command and control servers on April 1, 2009. At the time it was not known what would happen. Some thought it would be a massive attack, while others just thought the bots would connect for a simple wake up call. As it turns out, the bots in the botnet only connected for a simple wake up call. Although no massive worldwide attack has been recorded from the conflicker botnet there has been significant damage done because of its presence [99, 98, 46]. In conflicker there are approximately 50,000 possible domains which can serve as the command and control server which make it extremely difficult to discover the command and control since blacklisting a large list of non-static domain names is impractical [68]. The botnet authors also use asymmetric cryptographic authentication on the bots so the botnets cannot be reverse engineered by security professionals unless they have the private key. The botmasters that control conflicker are thought to have a high level of technical expertise. The malware writer for conflicker follows security research and updates the malware with the latest security technology. One such update was a security feature developed by M.I.T. researcher Ron Rivest. Conflicker added the security feature weeks after it was first introduced by Rivest, and when Rivest released a revision to correct a flaw in the feature the Conflicker authors updated the malware to reflect the revision [86] In late March, 2009 Felix Leder and Tillmann Werner from the Honeynet Project discovered a way to detect conflicker-infected hosts [44]. Other anti-virus companies have released tools to remove conflicker, but the botnet is still present on the Internet today. The command and control structure is mainly peer-to-peer.

7.2.1.3 Ongoing denial-of-service-attack of twitter

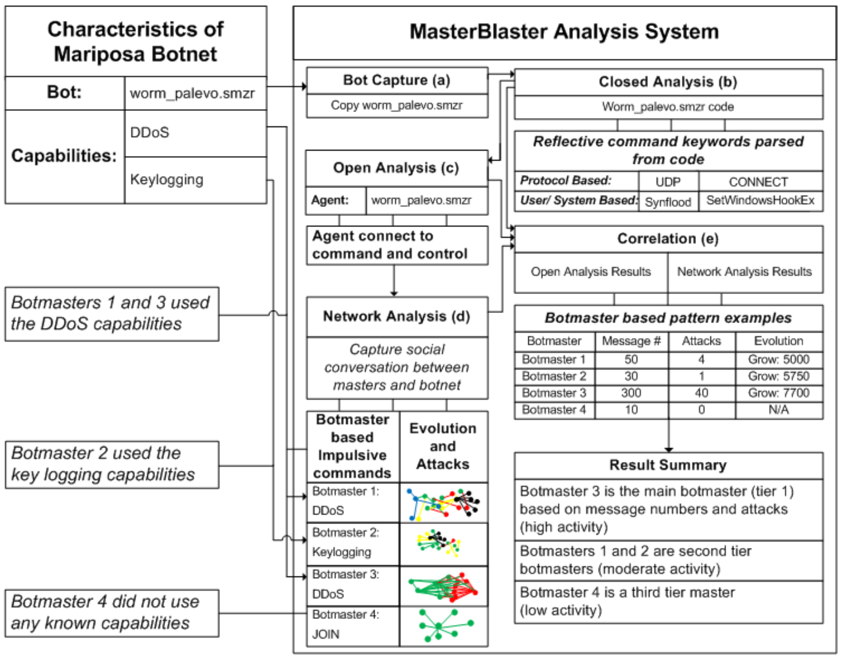
In August 2009 Twitter, Live Journal, YouTube, and Facebook were attacked by DDoS. The DDoS attack consisted of the botmaster infecting thousands of personal computers and then instructing all the computers to send view requests to each site. After further investigation it was discovered that the attacks were directed at one user that goes by the name Cyxymu. It was also discovered that the DDoS attack was related to massive spam that was sent out with Cyxymu's source address forged on them, which gave the appearance that all the spam originated from Cyxymu. The social sites of Cyxymu all contained substantial content promoting Georgia, where he is from. During this time there were tensions between Georgia and Russia, this gives us reason to believe that the attacks were Political in nature. Although the attacks were aimed at one user the result of the attacks shut down Twitter and Live Journal and also slowed down the services of Facebook and YouTube worldwide. This attack was carried out using the well known method of IRC for command and control communications. This is important to note, since research has seemingly shifted towards studying newer forms of botnet command and control communications. In fact there are still a large number of IRC controlled botnets still prevalent on the Internet today. The fact that the botnet responsible for this attack has not been shutdown and the botmasters have not been caught is evidence that IRC based botnets still need to be researched [101].

7.2.2 Using our method to analyze attacks

Each of the previously mentioned attacks has a different command and control structures. The Mariposa Botnet operates by having all the bots choose from a programmed list of DNS addresses to connect to and it communicates with the connected command and control using UDP. The Conflicker Botnet uses peer-to-peer communications to communicate and the Twitter attack uses IRC protocol like the botnet we analyze in the paper. While the monitoring methodology will change depending on protocol, the main analysis will remain the same. Instead of discovering the semantics of the commands using RFC 1459 and RFC 2812 for the IRC protocol, we would use RFC 5694 for peer-to-peer or a portion of the UDP protocol such as RFC 4113 for the Management Information Base for UDP. We would still employ our manual technique of monitoring the botnet for commands we do not find in the RFCs and discover meanings for them through our analysis. The rest of the process is the same as with the IRC protocol and the results will depend on the interactions of the botmasters and bots.

7.2.2.1 Mariposa botnet attack analysis

To get an idea of how our approach could work on a botnet with a command and control architecture other than IRC we explore possible results of analyzing the mariposa botnet. Figure 26 shows how we propose our framework will work on a botnet with a hybrid command and control architecture. In the figure we show the characteristics of a mariposa bot on the left side which includes the name of the infecting bot and two of the capabilities. On the right hand side we show how the components of our framework would process and analyze the botnet. In our example we see that there are three tiers of botmasters. To summarize this scenario; First the characteristics of the attack are modeled as behaviors using the reflective-impulsive method. Next the discovered behaviors are modeled using social networking evolution which evaluates the behavior based on the evolutionary phases the botnet goes through. Using these results we discover patterns to predict future actions. In figure 26, botmaster 3 generates many more transactions than any of the other botmasters. His


Characteristics of Mariposa Botnet
Bot:
worm_palevo.smzr
Capabilities:
DDoS
Keylogging
Botmasters 1 and 3 used the DDoS capabilities
Botmaster 2 used the key logging capabilities
Botmaster 4 did not use any known capabilities
MasterBlaster Analysis System
Bot Capture (a)
Copy worm_palevo.smzr
Closed Analysis (b)
Worm_palevo.smzr code
Reflective command keywords parsed from code
Protocol Based:
UDP
CONNECT
User/ System Based:
Synflood
SetWindowsHookEx
Open Analysis (c)
Agent:
worm_palevo.smzr
Agent connect to command and control
Correlation (e)
Open Analysis Results
Network Analysis Results
Network Analysis (d)
Capture social conversation between masters and botnet
Botmaster based pattern examples
Botmaster based Impulsive commands
Evolution and Attacks
Botmaster 1: DDoS
Botmaster 2: Keylogging
Botmaster 3: DDoS
Botmaster 4: JOIN
Result Summary
Botmaster 3 is the main botmaster (tier 1) based on message numbers and attacks (high activity)
Botmasters 1 and 2 are second tier botmasters (moderate activity)
Botmaster 4 is a third tier master (low activity)

Botmaster	Message #	Attacks	Evolution
Botmaster 1	50	4	Grow: 5000
Botmaster 2	30	1	Grow: 5750
Botmaster 3	300	40	Grow: 7700
Botmaster 4	10	0	N/A

Output from each component is input for next system component to discover botmaster based patterns

(a) A bot with file name "worm_palevo.smzr" is captured

(b) The bot code is parsed using the closed analysis system

- The parsed code is now split into two types of characteristics; Protocol Based commands and User/System Based commands

(c) The bot is sent to the open analysis module and turned into an agent under our control

- The agent is allowed to connect to the command and control center the bot was originally destined to connect to

(d) The transactions recorded by the agent is analyzed for conversations generated by bot masters

- Impulsive commands that are in-line with the bot capabilities discovered by the reflective command keywords are discovered
- The evolutionary stages and attacks are discovered: | Birth = Blue | Growth = Green | Contraction = Yellow | Death = Black | Attack = Red |

(e) Results from closed analysis component, open analysis component, and network analysis component are correlated

- Patterns are discovered and listed according to their botmaster name
- Results are conclusions made based on the patterns

Figure 26: The Mariposa Attack Modeled Using Our Framework

traffic is also much more attack laden which shows he is probably the most dangerous botmaster. Botmasters 1 and 2 do not generate a lot of transactions, but they both have an attack which took place so they are considered a moderate threat. Botmaster 4 did not generate much traffic at all, and no attacks took place. This botmaster is not considered a significant threat at the moment.

Although this was just an example loosely based on the characteristics of the mariposa botnet, an analysis would not likely differ greatly from what we have discussed. Within the mariposa botnet, thousands of companies were attacked by thousands of botmasters. By identifying what botmaster conducted the attack we can create a list of suspects to help toward possible attribution. Using the previous example in figure 26, if we are examining a DDoS attack we can eliminate botmasters 2 and 4 as suspects since they did not conduct any DDoS related activities. Furthermore we can identify Botmaster 3 as the lead suspect since he had the most DDoS related messages.

CHAPTER 8: CONCLUSION

The real threat from botnets come from the botmasters that use them. Since multiple botmasters are found on each separate botnet channel, automatically discovering the role each botmaster plays will significantly reduce analysis time and provide a list of suspects to aid in the attribution of attacks. We believe the results we have discovered are an important and encouraging start to identifying the patterns created by the transactions of botmasters and future work could prove invaluable to identify and defend against the persistent threat of botnets. Our contributions are as follows:

- **Identified the importance of botmaster based analysis:** Current botnet analysis methodologies have focused on either discovering bots in botnets or studying the magnitude of the botnet problem. Although these focuses are necessary, the botnet problem continues to worsen. In our work we focus on a more fine grain analysis. We first address a deficiency present in most botnet diagrams by including the fact that most botnets are controlled by multiple botmasters instead of just one. We then discovered that each botmaster has their own agenda which can be identified through the analysis of the commands they submit to nodes within the botnet channels along with the capabilities of the bots that reside on the botnet. Our results prove that multiple botmasters on the same botnet do in fact have different agendas, which is especially evident by the different types of attacks committed by botmasters on the same botnet. To the best of our knowledge this is the first attempt to study botnets based on the fine-grained analysis of the botmaster.
- **Introduced characteristic discovery through bot code contents:** Previous research in bot analysis has focused on shutting down botnets based on the types of bots it uses, or identifying the botnet based on the bots. In our work we introduce the ability to discover meaningful characteristics by matching keywords found in the bot code with commands the botmaster sends to other bot nodes within the botnet. Using this methodology we were able to identify characteristics and the implied motives of the botmasters.
- **Identified the importance of evolution study in botnet analysis:** We showed that our hypothesis which states (most botnet attacks will occur while

the botnets are at their largest), is most likely true. More testing over longer periods of time need to be conducted to definitively prove this is true. We also discovered that larger botnets are more dynamic than smaller botnets in relation to botnet nodes entering and leaving the botnet channels. Larger botnets tend to recycle botnets at a much faster rate than smaller botnets. This is a very interesting trend, especially since other networks have been proven to have the same attributes. To the best of our knowledge this is the first time evolution of a botnet has been studied.

We believe our research has opened the door for many research projects in the future. Some of the future projects we plan to work on include:

- **Botnet based analysis on multiple command and control protocols:** Our current research was on IRC based botnets only. Although IRC based botnets still present a significant threat, other forms of botnets have emerged as more difficult to stop due to their command and control architectures. Theoretically our approach should be able to work on any command and control since we determine the command structure based on the protocols. In the future we plan on extending our approach to these other botnets.
- **Fine-grained analysis of bot code for characteristic discovery:** Our current approach was sufficient in identifying keywords that could match the commands of the botmasters. In the future we will be doing a through malware analysis of the bot code to discover what commands are possible with the malware. We will then match the bot command capabilities with the commands sent from the botmasters. This will give us a better idea of the skill level of the botmasters and it will give us a more detailed view of the botmaster characteristics and motives.
- **Hardware based DFA implementation of data collection:** Our software based solution worked well for our approach, but in order to create an even more efficient approach we will investigate hardware based solutions. A more efficient approach will give us the flexibility to place our system in-line instead of off-line.
- **Autonomous Protection Mechanism:** Currently our protection component consists of a web-based front-end that is generated by the administrator. In the future we will research more automatic means to protecting systems from botmasters. One approach we will explore will be to automatically block traffic and send alerts to administrators when the threat level reaches a certain level for a certain botmaster.

In conclusion we believe the results we have discovered has made significant contributions to the field of network security. Botmasters today operate under the protection and comfort of anonymity. By identifying attributes of botmasters, we move a step closer to attribution of botnet attacks. We believe that the threat of attribution will reduce the amount of botmasters that carryout attacks using botnets and it is our goal that future research in botmaster based analysis will reduce the impact of botnet attacks in the near future.

REFERENCES

[1] Cyber protests: The threat to the u.s. information infrastructure. *www.au.af.mil/au/awc/awcgate/nipc/cvberprotests.pdf* (April 2001), 1.

[2] Abu Rajab, M., Zarfoss, J., Monrose, F., and Terzis, A. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement* (New York, NY, USA, 2006), ACM, pp. 41–52.

[3] Adair, S. The website for the president of georgia under attack - politically motivated? *www.shadowserver.org/wiki/pmwiki.php* (April 2008), 1.

[4] Ahn, G., Paxton, N. C., and Pearson, K. Understanding irc bot behaviors in network-centric attack detection and prevention framework. In *ICIW 2008* (2008).

[5] Alto, P. Mariposa how exposed are we. *www.paloaltonetworks.com/mariposa/*2009 (April 2009), 1.

[6] Baecher, P., Kotter, M., Holz, T., Dornseif, M., and Freiling, F. The nepenthes platform: An efficient approach to collect malware. In *RAID '06: Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection* (2006), pp. 41–52.

[7] Barford, P., and Yegneswaran, V. An inside look at botnets. In *Special Workshop on Malware Detection.*

[8] Binkley, J. R. Anomaly-based botnet server detection. In *FloCon 2006.*

[9] Binkley, J. R., and Singh, S. An algorithm for anomaly-based botnet detection. In *USENIX SRUTI.*

[10] Caballero, J., Yin, H., Liang, Z., and Song, D. Polyglot: Automatic extraction of protocol message format using dynamic binary analysis. In *the 14th ACM Conference on Computer and Communications Security* (2007).

[11] Cai, J. Honeynets and honeygames: A gametheoretic approach to defending network monitors. In *Tech Rep TR1577* (2006).

[12] Calvet, J., Davis, C., and Pierre-Marc, B. Malware authors don't learn, and that's good. In *MALWARE'09: Proceedings of the Fourth Annual Conference on Malicious and Unwanted Software* (2009), IEEE.

[13] Carr, J. Why i believe that the kyrgyzstan government hired russian hackers to launch a ddos attack against itself. *http://intelfusion.net/wordpress/?p*520 (April 2009), 1.

[14] Carr, J., Walton, G., and Walton, A. The kyrgyzstan ddos attacks of january, 2009: Assessment and analysis. *http : //intelfusion.net/wordpress/?p* = 516 (April 2009), 1.

[15] Chia, G., Barabasi, A.-L., and Vicsek, T. Quantifying social group evolution. *Nature 446* (April 2007), 664–667.

[16] Cho, C. Y., Babic, D., Shin, E. C. R., and Song, D. Inference and analyis of formal models of botnet command and control protocols. In *the 17th ACM Conference on Computer and Communications Security* (2010).

[17] Clamav. clamav. *www.clamav.net* (April 2007), 1.

[18] cnet. Bots slim down to get tough. *http : //news.com/..tough/*213*.html* (April 2005), 1.

[19] Corrons, L. Mariposa botnet. *http : //pandalabs.pandasecurity.com/mariposa − botnet* (March 2010), 5.

[20] cup. cup. *http : //www*2*.cs.tum.edu/cup* (January 2010), 1.

[21] Dagon, D., Gu, G., Zou, C., Grizzard, J., Dwivedi, S., Lee, W., and Lipton, R. A taxonomy of botnets. In *the 23rd Annual Computer Security Applications Conference* (2007).

[22] Denning, D. E. Activism, hactivism, and cyberterrorism: The internet as a tool for influencing. In *Networks and Netwars: The future of terror, crime, and militancy* (2001).

[23] Dunn, J. Mcafee launches first bot-killing system. *www.techworld.com* (April 2007), 1.

[24] eWEEK. Money bots: Hackers cash in on hijacked pcs. *www.eweek.com/article*2/0.957.00*.asp* (April 2006), 1.

[25] Freiling, F., Holz, T., and Wicherski, G. Botnet tracking: Exploring a root-cause methodology to prevent denial of service attacks. In *ESORICS'05: Proceedings of ESORICS 2005* (2005), IEEE.

[26] Geers, K. Sun tzu and cyber war. *www.ccdcoe.org/articles/CyberWar.pdf* (April 2011), 1.

[27] Gu, G., Yegneswaran, V., Porras, P., Stoll, J., and Lee, W. Active botnet probing to identify obscure command and control channels. In *ACSAC 09: Proceedings of the 2009 Annual Computer Security Applications Conference* (Washington, D.C., 2009), IEEE, pp. 241–253.

[28] Holz, T. A short visit to the bot zoo. In *Security and Privacy Magazine* (2005).

[29] HONEYNET. Know your enemy: Tracking botnets. *http : //www.honeynet.org/papers/bots* (August 2008).

[30] HOULE, K., AND WEAVER, G. Trends in denail of service attack technology. In *CERT* (2001).

[31] HUNKER, J., HUTCHINSON, B., AND MARGULIES, J. Role and challenges for sufficient cyber-attack attribution. *www.thei3p.org/whitepaper − attribution.pdf* (April 2008), 1.

[32] IC3. 2009 internet crime report. *http : //www.ic3.gov/media/annualreport/2009$_I$C3Report.pdf* (March 2010).

[33] INTELLIGENCE, D. Mariposa botnet analysis. In *Technical Report* (2009).

[34] JACKSON, D. Kyrgyzstan under ddos attack from russia. *www.secureworks.comkyrgyzstan − under − ddos − attack − from − russia* (April 2009), 1.

[35] JFLEX. Jflex. *http : //jflex.de* (January 2010), 1.

[36] KARASARIDIS, A., REXROAD, B., AND HOEFLIN, D. Wide-scale botnet detection and characterization. In *Usenix hotbots 2007* (2007).

[37] KARGL, F., AND WEBER, M. Protecting webservers from distributed denial of service attacks. In *the 10th International World Wide Web Conference* (2001).

[38] KIRK, J. Student fined for attack against estonian website. 1.

[39] KREBS, B. Storm worm dwarfs worlds top supercomputers.

[40] KRISTOFF, J. Botnets. In *NANOG32* (2004).

[41] KUECHLER, W., AND VAISHNAVI, V. Design [science] research in is: A work in progress. In *the 2nd International Conference on Design Science Research in Information Systems and Technology* (2007).

[42] LAU, F., RUBIN, H., SMITH, M., AND TRAJKOVIC, L. Distributed denial of service attacks. In *international Conference on Systems, Man, and Cybernetics* (2000).

[43] LEDER, F., AND WERNER, T. Know your enemy: Containing conflicker. *The Honeynet Project* (April 2009).

[44] LEDER, F., WERNER, T., AND MARTINI, P. Proactive botnet countermeasures - an offensive approach. In *In Cooperative Cyber Defense Centre of Excellence* (Tallinn, Estonia, 2009).

[45] LEWIS, J. Assessing the risks of cyber terrorism, cyber war and other cyber threats:. *www.steptoe.com/publications/231a.pdf* (April 2002), 1.

[46] Leyden, J. Conflicker seizes city's hospital network. *The Register* (July 2009).

[47] Li, J., Ehrenkranz, T., and Kuenning, G. Simulation and analysis on the resiliency and efficiency of malnets. In *the 19th Workshop on Principles of Advanced and Distributed Simulation* (2005).

[48] Li, Z., Goyal, A., and Chen, Y. Honeynet-based botnet scan traffic analysis.

[49] Li, Z., Goyal, A., Chen, Y., and Paxson, V. Automationg analysis of large-scale botnet probing events. In *ASIACCS 09* (2009).

[50] Livadas, C., Walsh, R., Lapsley, D., and Strayer, W. T. Using machine learning techniques to identify botnet traffic. In *2nd IEEE LCN workshop on Network Security*.

[51] m86security. Security labs report. $http://www.m86security.com/M86_{L}abs_{R}eport_{J}an2010.pdf$ (January 2010), 5.

[52] Mao, C., Chen, Y., Huang, S., and Lee, H. Irc-botnet network behavior detection in command and control phase based on sequential temporal analysis. In *CISC 2009*.

[53] maxmind. maxmind. *www.maxmind.com* (April 2007), 1.

[54] McLaughlin, L. Bot software spreads, causes new worries. $http://csdl2.computer.org/comp/mags.pdf$ (April 2004), 1.

[55] Mirkovic, J. Distributed denial-0f-service attacks. In *PhD thesis* (2003).

[56] mwcollect. Nepenthes-finest collection. *www.nepenthes.mwcollect.org* (April 2007), 1.

[57] MySql. Mysql. *www.mysql.com* (April 2007), 1.

[58] Namestnikov, Y. The economics of botnets. $http://www.securelist.com/ynambotnets0907en.pdf$ (July 2009), 11.

[59] Nazario, J. Estonian ddos attacks - a summary to date. $http://asert.arbornetworks.com/estonian-ddos-attacks-a-summary-to-date$ (May 2007), 1.

[60] Nazario, J. Politically motivated denial of service attacks. $www.ccdcoe.org/publications/virtualbattlefield$ (April 2007), 1.

[61] Nazario, J. Cnn attack summary. $http://asert.arbornetworks.com/cnn-attack-summary$ (April 2008), 1.

[62] Nazario, J. The effects of war: Gaza and isreal. $http://asertarbornetworks.com/the-effects-of-war-gaza-and-israel$ (April 2009), 1.

[63] Nazario, J., and Dimino, A. M. An in-depth look at the georgia-russia cyber conflict of 2008. In *the Botnet Task Force Meeting* (2008).

[64] News, F. Estonia hosts georgian websites to halt hackers. *www.foxnews.com/wires/EstoniaGeorgiaHaltingHackers00.html* (April 2008), 1.

[65] Nunnery, C., Sinclair, G., and Kang, B. B. Tumbling down the rabbit hole: Exploring the idiosyncrasies of botmaster systems in a multi-tier botnet infrastructure. In *LEET'10: Proceedings of 3rd Annual Workshop on Large-Scale Exploits and Emergent Threats* (San Jose, CA, 2010), USENIX.

[66] Palla, G., Barabasi, A.-L., and Vicsek, T. Quantifying social group evolution. *Nature 446* (April 2007), 664–667.

[67] Park, K., Pai, V. S., Lee, W. K., and Calo, S. Securing web service by automatic robot detection. In *USENIX 2006* (2006).

[68] Park, Y., and Reeves, D. S. Identification of bot commands by run-time execution monitoring. In *ACSAC '09: Proceedings of the 2009 Annual Computer Security Applications Conference* (Washington, D.C., 2009), IEEE, pp. 321–330.

[69] Park, Y. H., and Reeves, D. S. Adaptive timing-based active watermarking for attack attribution through stepping stones. *ftp : //ftp.eos.ncsu.edu/pub/tech.pdf* (April 2007), 1.

[70] Paxton, N. C., Ahn, G., and Chu, B. Towards practical framework for collecting and analyzing network-centric attacks. In *IRI'07: Proceedings of the IEEE International Conference on Information Reuse and Integration* (Las Vegas, NV, 2007), IEEE, pp. 73–78.

[71] Paxton, N. C., Ahn, G., Kelly, R., Pearson, K., and Chu, B. Collecting and analyzing bots in a systematic honeynet-based testbed environment. In *the 11th Colloquium for Information Systems Security Education* (2007).

[72] Peng, P., Ning, P., and Reeves, D. S. On the secrecy of timing-based active watermarking trace-back techniques. In *2006 IEEE Symposium on Security and Privacy* (2006).

[73] perileyez. Perileyez. *www.digitalninjitsu.com/downloads.html* (April 2007), 1.

[74] Project, H. Know your enemy: Sebek- a kernel based data capture tool. *www.honeynet.org/papers* (April 2003), 1.

[75] Project, H. Know your enemy: Genii honeynets. *www.honeynet.org/papers* (April 2006), 1.

[76] PROJECT, H. Know your enemy: Tracking botnets. *www.honeynet.org/papers* (April 2007), 1.

[77] PROVOS, N. A virtual honeypot framework. In *USENIX Security 2004* (2004).

[78] RACINE, S. Analysis of internet relay chat usage of ddos zombies. In *Master's thesis, ETH* (2004).

[79] RAMACHANDRAN, A., FEAMSTER, M., AND DAGON, D. Revealing botnet membership using dnsbl counter-intelligence. In *2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet* (2006).

[80] RUBENKING, N. J. New botnet may have infected half of fortune 1000. *PC-MAG.COM* (August 2009).

[81] RUSSIA, T. O. Russian opposition websites shut down by attacks. *www.theotherrussia.org/russian – opposition – websites – shut – down – by – attacks* (April 2008), 1.

[82] SINHA, P., BOUKHTOUTA, A., BELARDE, V. H., AND DEBBABI, M. Insights from the analysis of the mariposa botnet. In *5th International Conference on Risks and Security of Internet and Systems* (2010).

[83] SONG, J., KIM, J., SEO, D., SOH, W., AND KIM, S. Something about dfas. In *Communications in Computer and Information Science 2010* (2010).

[84] SONG, J., KIM, J., SEO, D., SOH, W., AND KIM, S. Study of host-based cyber attack precursor symptom detection algorithm. In *Communications in Computer and Information Science 2010* (2010).

[85] SPATSCHECK, O., AND PETERSEN, L. L. Defending against denial of service attacks in scout. In *the 3rd Symposium of Operating Systems Design and Implementation* (1999).

[86] STAFF, R. Experts team up to battle conflicker botnet. *RedOrbit* (March 2009).

[87] STANIFORD, S., PAXSON, V., AND WEAVER, N. How to 0wn the internet in your spare time. In *the 11th USENIX Security Symposium* (2002).

[88] STANIFORD-CHEN, S., AND HEBERLEIN, L. T. Holding intruders accountable on the internet. In *the 1995 IEEE Symposium on Security and Privacy* (1995).

[89] STEWARD, J. Truman - the reusable unknown malware analysis net. *www.lurhq.com/truman* (April 1991), 1.

[90] STINSON, E., AND MICHELL, J. C. Characterizing bots remote control behavior. In *GI SIG SIDAR conference on detection of intrusion and malware vulnerability assessment.*

[91] Strack, F., and Deutsch, R. Reflective and impulsive determinants of social behavior. *In Proceedings of Person. Soc. Psychol. Rev 8 8* (March 2004), 220–247.

[92] Strayer, W. T., Lapsley, D., Walsh, R., and Livadas, C. Botnet detection based on network behavior. In *Springer-Verlag 2008.*

[93] Symantec. symantec. *www.symantec.com* (April 2007), 1.

[94] Tikk, E., Kaska, K., Runnimeri, K., Kert, M., and Taliharm, A. Cyber attack against georgia: Legal lessons identified. 1.

[95] Truman. Truman-the reusable unknown malware analysis net. *www.lurhq.com/truman/* (April 2007), 1.

[96] VMware. Vmware gsx server. *www.vmware.com* (April 2007), 1.

[97] Wang, X., Reeves, D. S., Ning, P., and Feng, F. Network-based attack attribution through probabilistic watermarking of packet flows. In *Report TR-2005-10* (2005).

[98] Williams, C. Conflicker seizes city's hospital network. *The Register* (Janurary 2009).

[99] Wilsher, K. French fighter planes grounded by computer virus. *Telegraph.co.uk* (February 2009).

[100] wireshark. wireshark. *www.wireshark.com* (April 2007), 1.

[101] Wortham, J., and Kramer, A. E. Professor main target of assault on twitter. *The New York Times* (August 2009), B1.

[102] Yegneswaran, V., Barford, P., and Paxson, V. Using honeynets for internet situational awareness. In *ACM Hotnets IV* (2005).

[103] Zheng, Y. L., and Leiwo, J. D-ward: Source-end defense against distributed denial of service protection base. In *Information Security and Privacy* (1997).

[104] Zhichun, L., Goyal, A., Chen, Y., and Paxson, V. Towards situational awareness of large-scale botnet probing events. In *IEEE Transactions on Information Forensics and Security.*

[105] Zou, C., Gao, L., Gong, W., and Towsley, D. Monitoring and early warning for internet worms. In *ACM CCS* (2003).