The Next Generation Botnet Attacks And Defenses

by

PING WANG
M.S. Beijing University of Aeronautics and Astronautics, China, 2004
B.S. Beijing University of Aeronautics and Astronautics, China, 2001

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2010

Major Professor: Cliff C. Zou

ABSTRACT

A "botnet" is a network of compromised computers (bots) that are controlled by an attacker (botmasters). Botnets are one of the most serious threats to today's Internet; they are the root cause of many current Internet attacks, such as email spam, distributed denial of service (DDoS) attacks , click fraud, etc. There have been many researches on how to detect, monitor, and defend against botnets that have appeared and their attack techniques. However, it is equally important for us to investigate possible attack techniques that could be used by the next generation botnets, and develop effective defense techniques accordingly in order to be well prepared for future botnet attacks.

In this dissertation, we focus on two areas of the next generation botnet attacks and defenses: the peer-to-peer (P2P) structured botnets and the possible honeypot detection techniques used by future botnets.

Currently, most botnets have centralized command and control (C&C) architecture. However, P2P structured botnets have gradually emerged as a new advanced form of botnets. Without C&C servers, P2P botnets are more resilient to defense countermeasures than traditional centralized botnets.

Therefore, we first systematically study P2P botnets along multiple dimensions: bot candidate selection, network construction and C&C mechanisms and communication protocols.

As a further illustration of P2P botnets, we then present the design of an advanced hybrid P2P botnet, which could be developed by botmasters in the near future. Compared with current botnets, the proposed botnet is harder to be shut down, monitored, and hijacked. It provides robust network connectivity, individualized encryption and control traffic dispersion, limited botnet exposure by each bot, and easy monitoring and recovery by its botmaster. We suggest and analyze several possible defenses against this advanced botnet.

Upon our understanding of P2P botnets, we turn our focus to P2P botnet countermeasures. We provide mathematical analysis of two P2P botnet mitigation approaches — index

poisoning defense and Sybil defense, and one monitoring technique - passive monitoring. We are able to give analytical results to evaluate their performance. And simulation-based experiments show that our analysis is accurate.

Besides P2P botnets, we investigate honeypot-aware botnets as well. This is because honeypot techniques have been widely used in botnet defense systems, botmasters will have to find ways to detect honeypots in order to protect and secure their botnets. We point out a general honeypot-aware principle, that is security professionals deploying honeypots have liability constraint such that they cannot allow their honeypots to participate in real attacks that could cause damage to others, while attackers do not need to follow this constraint. Based on this principle, a hardware- and software- independent honeypot detection methodology is proposed. We present possible honeypot detection techniques that can be used in both centralized botnets and P2P botnets. Our experiments show that current standard honeypot and honeynet programs are vulnerable to the proposed honeypot detection techniques. In the meantime, we discuss some guidelines for defending against general honeypot-aware botnet attacks.

Dedicated to my family.

ACKNOWLEDGMENTS

First, I would like to thank my advisor, Dr. Cliff Zou, for his invaluable guidance, encouragement and generous support throughout my time in the Ph.D. program at University of Central Florida. He helped me build my self-confidence, such that today I am able to successfully finish my Ph.D study.

Second, I would like to thank three of my dissertation committee members, Dr. Ratan Guha, Dr. Damla Turgut and Dr. Morgan Wang, for their precious time and valuable comments and suggestions on my dissertation.

And I would also like to thank Dr. Ning Zhang for his financial support. Because of that, I was able to focus on my research without worrying about my daily life.

I would especially like to give thanks to my dear parents and my dear husband for always being there for me. Their wholehearted support has always been keeping me strong and helping me get through difficult times.

Last but not the least, I would like to thank all my friends here in the U.S. and those back in China. Over the past years, they keep me happy and make my life more colorful.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Motivations

A "botnet" is a network of compromised computers ("bots") running malicious software, usually installed via all kinds of attacking techniques such as Trojan horses, worms and viruses. These zombie computers are remotely under the control of a single entity ("botmaster"). Botnets with a large number of computers have enormous cumulative bandwidth and powerful computing capability. They are exploited by botmasters for initiating various malicious activities, such as email spam, click fraud, distributed denial-of-service (DDOS) attacks, password cracking, key logging and etc. And now botnets are the main cause of these attacks.

Back in 2007, between 100 million and 150 million of the 600 million PCs on the Internet are under the control of hackers [1]. As of 2006, researchers have put the average botnet size around 20,000 compromised machines [89]. Later on in 2007, it was shown that the size of botnet like Storm botnet ranged from 1 to millions [62] and in 2008 concurrent online Storm nodes [64] were between 5,000 and 40,000 [63]. According to a report from Symantec's MessageLabs, only in June 2009, 90.4% of all e-mails are spam, however, about 83.2% of spam are sent by botnets [25]. Botnets have become one of the most significant threats to the Internet.

There have been many researches on botnet detection, monitoring and shutdown, such as [50, 43, 97, 32, 58]. However, they focus mainly on botnets that have appeared in the past several years, studying the behaviors of those botnets and developing corresponding detection, monitoring and shutdown techniques. Such research is essential and critical for building our defense systems against current botnets, but they may become ineffective if botmasters advance the attack techniques to create new generation botnets. Without looking

forward, we would always stay one step behind botmasters. However, it is equally important for us to investigate possible attack techniques that could be used by the next generation botnets, and develop effective defense techniques accordingly in order to be well prepared for future botnet attacks.

In this dissertation, we focus on two areas of the next generation botnet attacks and defenses: the peer-to-peer (P2P) structured botnets and the possible honeypot detection techniques used by future botnets.

1.2 Proposed Work and Contributions

In this section, we give an overview of the proposed work on the next generation botnet attacks and defenses. Before the detailed discussion, we would like to summarize our major contributions, which are listed as follows.

- We systematically study P2P botnets along multiple dimensions: infection vectors, bot candidate selection, bootstrap procedure, network structure, C&C mechanisms and communication protocols [125].

- An advanced hybrid peer-to-peer botnet is presented [124]. Compared with current botnets, the proposed one is much harder to be shut down or monitored. It provides robust network connectivity, individualized encryption and control traffic dispersion, limited botnet exposure by each captured bot, and easy monitoring and recovery by its botmaster.

- We present a number of countermeasures to defend against P2P botnets [125]. Our study on index poisoning defense and Sybil defense shows that both of defense approaches share a common scheme, which leads us to analytical work on the evaluation of their performance. In addition, we address the effectiveness of passive monitoring of

P2P botnets by providing a lower bound for the number of bots that a node controlled by defenders can monitor.

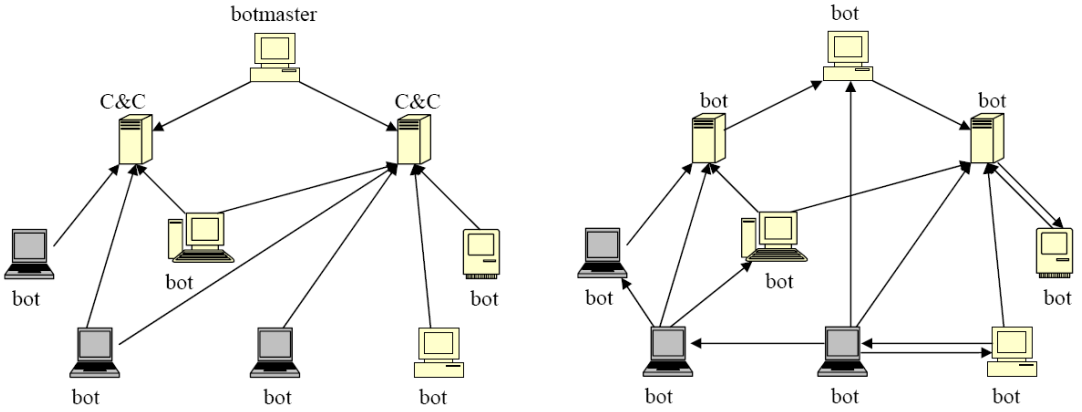
- We develop a Kademlia-based P2P botnet simulator. All the analytical work on P2P botnet mitigation presented in this dissertation have been shown to be accurate by simulation-based experiments.

- For honeypot-aware botnets, we introduce various means by which attackers could detect honeypots in their constructed botnets based on a principle which has to be complied by defenders but not attackers [126]. On the other hand, we discuss possible directions for defenses against such honeypot-aware botnets.

1.2.1 Peer-to-Peer Botnets

Most botnets that have appeared until now have a common centralized architecture. That is, bots in the botnet connect directly to some special hosts, which are referred as "*command-and-control*" (C&C) servers. These C&C servers receive commands from their botmaster and forward them to the other bots in the network. We call a botnet with such a control communication architecture a "centralized botnet". Fig. 1.1(a) shows the basic control communication architecture for a typical centralized botnet (in reality, a centralized botnet usually has more than two C&C servers). Arrows represent the directions of network connections.

Today, centralized botnets are still widely used. Among them, Internet relay chat (IRC)-based botnets [132] are the most popular ones, which use IRC [68] to facilitate command and control communication between bots and botmasters. This centralized architecture is easy to construct and very efficient in distributing botmaster's commands; however, it has a single point of failure - the C&C server. Shutting down the IRC server would cause all the bots lose contact with their botmaster. In addition, defenders can also easily monitor the botnet by creating a decoy to join the specified IRC channel.



(a). Centralized Botnet (b). P2P Botnet

Figure 1.1: Two botnet architectures

Recently, peer-to-peer (P2P) botnets, such as Trojan.Peacomm botnet, Storm botnet and its newly improved version Waledac botnet, have emerged, as attackers gradually realize the limitations of traditional centralized botnets. "Peer-to-peer botnets" are defined as botnets that rely on peer-to-peer communication mechanisms to facilitate the command and control by their botmasters. There are different ways for a P2P botnet to utilize P2P communication for its command and control. For example, a P2P botnet could use a P2P network (either an existing P2P network, or a unique P2P network formed by its bot members) to directly disseminate its botmaster's commands to all bot members, or it could use a P2P network to disseminate the IP addresses of C&C servers to bot members (like what Storm botnet [110] does, which utilizes an existing P2P protocol to form a hierarchical multi-tier command and control architecture). Due to the fundamental distributive nature of P2P networks, P2P botnets are robust against removal of bots and C&C servers, and have shown great advantages over traditional centralized botnets. As the next generation of botnets, they are more robust and difficult for security community to defend against.

Researchers have started to pay attention to P2P botnets. Trojan.Peacomm botnet, Stormnet and Waledac botnet have been detailedly dissected in literature [57, 64, 91, 53, 111, 88]. However, in order to effectively fight against this new form of botnets, enumerating

every individual P2P botnet we have seen in the wild is not enough. Instead, we need to study P2P botnets in a systematic way.

Therefore in this dissertation we aim to study P2P botnets from three aspects and address the following questions:

P2P Botnets Themselves How is a P2P botnet is constructed? And how do bot members communicate with each other and their botmaster?

From Attackers' Perspective How to build a P2P botnet and make it easy to control and more robust and resilient to defense techniques?

From Defenders' Perspective How to mitigate P2P botnets? And how to evaluate the P2P botnet countermeasures and make them more effective?

First we start with analyzing P2P botnets. We consider the life cycle of a P2P botnet is a three-stage process, which is composed of bot selection, network construction and botnet operation (e.g. C&C communication). During each stage, different P2P botnets may use different mechanisms. We present most common mechanisms from multiple aspects, such as the design details, their advantages and limitations, etc.

To further illustrate our study on P2P botnets, we present an advanced hybrid P2P botnet. We eliminate the bootstrap procedure, which could be a single point of failure during botnet construction. Each bot only knows limited number of bot members, and the encrypted commands are passed by the ones with stable and static IP addresses. These features improve the botnet's resilience against infiltration. Moreover, we find a way to make the botnet more robust by balancing the connections among bots. In one word, the design of this botnet overcomes some of the challenges faced by botmasters, which will be discussed in the later chapter, and give us a good understanding of how a future P2P botnet might be.

However, our ultimate goal is to be able to better fight against P2P botnets, a new generation of botnets. Therefore we study possible P2P botnet countermeasures, such as

botnet detection, monitoring, and shutdown. Because of the similarities shared by all botnets, some techniques which are applied to traditional centralized botnets can also be used for P2P botnet mitigation with modifications, e.g. Holz *et al.* [64] adapted the technique of botnet tracking to study Storm botnet, which is a P2P based botnet. On the other hand, P2P botnets have their own features which could be leveraged by defenders. For example, index poisoning defense and Sybil defense are very effective techniques aiming at a certain type of P2P botnets, such as Stormnet and Trajon.Peacomm, by taking advantage of the specific C&C communication design of this type of P2P botnets. We give detailed studies on index poisoning defense and Sybil defense techniques, present analytical work on evaluation of their performance. Furthermore, passive monitoring as an effective method to capture bots in P2P botnet is discussed and analyzed as well.

1.2.2 Honeypot Detection

"Honeypot" is a decoy computer system putting on a network as bait for attackers to attack [65]. In recent years, honeypots have become popular and been widely studied by security researchers and industry. Many successful honeypot-based attack analysis and detection systems have been designed, such as [45, 93, 67]. As more people deploy honeypots in botnet monitoring and defense systems, botmasters will sooner or later modify their bot codes to find ways to avoid honeypot traps. Therefore, we should study honeypot avoidance techniques that could be deployed by botmasters in their next generation botnets in order for us to design better and long-lasting honeypot-based defense systems.

Current honeypot detection techniques discussed in the literature are usually based on hardware or software behavior differences between honeypots and real machines, e.g., by detecting VMware or other emulated virtual environments [102, 44], or by detecting the specific honeypot program's faulty responses [21]. However, these honeypot detection techniques target narrowly to specific honeypot configurations; they are not suitable for botnets

since botnets find and compromise computers automatically and could face with any type of honeypot configurations.

For these reasons, botmasters will develop and deploy a general-purpose honeypot detection technique in their next generation botnets. We propose and investigate such a general-purpose honeypot detection principle: security professionals deploying honeypots have liability constraint, either legally or ethnically, such that they cannot allow their honeypots to participate in real attacks that could cause damage to others, while attackers do not have this constraint. Botmasters could detect honeypots in their botnets by checking whether compromised machines in a botnet can successfully send out unmodified malicious traffic. Based on this principle, we present honeypot detection techniques which work in both centralized botnets and P2P botnets.

1.3 Organization of the Dissertation

The remainder of the dissertation is organized as follows. Chapter 2 provides background knowledge and literature review in related areas. Chapter 3 gives a detailed systematic study on P2P botnets. An advanced hybrid P2P botnet is proposed in Chapter 4. Then we present possible P2P botnet countermeasures and especially study index poisoning defense, Sybil defense and passive monitoring techniques in Chapter 5. In Chapter 6, we focus on honeypot-aware botnets, and propose a honeypot detection methodology with detailed discussion. Finally, Chapter 7 concludes the dissertation and discusses our future work.

CHAPTER 2: BACKGROUND AND RELATED WORK

The technology to remotely control compromised computers first surfaced in late 1999 when SANS Institute researchers discovered remotely executable code on thousands of Windows machines [94]. Since then, more and more botnets start to emerge, such as Slapper [28], Sinit [15], Phatbot [12], Nugache [77], Kraken botnet [7], etc. Among them, P2P botnets, like Trojan.Peacomm botnet [57] and Stormnet [64], appeared in recent years to be a new form of botnets. Accordingly, security researchers and professionals spend lots of effort on botnet defense. And honeypot-based attack analysis and detection systems have shown to be successful. So it is highly possible that attackers will promote botnets with the ability to detect honeypots. Peer-to-peer botnets and honeypot-aware botnets are the key points of this dissertation, we will give some background knowledge and a review of recent related work on peer-to-peer networks, botnets and honeypot techniques in this chapter.

2.1 Peer-to-Peer Networks

A P2P network is a computer network in which two or more computers are connected and share resources (content, storage, CPU cycles) by direct exchange, rather than going to a server or authority which manages centralized resources [27]. Today one of the major uses of P2P networks is file-sharing, and there are various P2P file-sharing applications, such as eMule [5], Lime Wire [8], BitTorrent [2]. P2P networks can be distinguished in terms of their centralization and structure [27].

Speaking of overlay network centralization, there are two categories (Table 2.1).

Centralized Architectures In this class of P2P networks, there is a central server which maintains directories of metadata and routing information, processes file search requests and coordinates download among peers. Napster [10] was the first widely-used P2P sharing application, which used this centralized architecture. It had a central site, which receives

search, browse or file transfer requests sent by peers, and sends responses back to them. But the central site does not participate in actual file downloading. Obviously this central site is a single point of failure and limits network scalability and robustness.

Decentralized Architectures In this class of P2P networks there is no central server. All requests are handled by peers within the network. In purely decentralized architectures, each peer behaviors exactly the same. They act as a server when processing a file search query, and as a client when requesting a file. However in partially decentralized architectures, peers with better computing ability and network bandwidth have the chance to be promoted and become "super peers" playing a more important role in the network. Gnutella network [6] is a partially decentralized P2P network. Normal peers (so- called leaf peers) in Gnutella network is connected to at least one super peer (so-called ultrapeer) and can only send queries to its ultrapeers. An ultrapeer maintains a table of hash values of files which are available in its local leaf peers. Only ultrapeers can forward messages. Therefore, from the leaf peer's perspective, an ultrapeer acts as a local server.

Table 2.1: Classification based on centralization

Category		**Network or Protocol**	**Applications**
Centralized		Napster	Napster
Decentralized	Purely	Gnutella (before 2001)	LimeWire
	Partially	Gnutella (after 2001) FastTrack	LimeWire Kazaa

Table 2.2: Classification based on structure

Category	**Network or Protocol**	**Applications**
Structured	Overnet	eDonkey2000, eMule
Unstructured	Gnutella	LimeWire

P2P networks can also be classified as either unstructured or structured networks (Table 2.2).

Unstructured In unstructured P2P networks, content location is completely unrelated to the network topology. A query usually flooded throughout the network, or forwarded in depth-first or breadth-first manner, until a query hit is reached or queries expire. Gnutella network is unstructured. Ultrapeers forward queries to its neighboring ultrapeers, and the query hit will be sent back along the same route as the search query.

Structured In structured P2P networks, a mapping is created between the content (e.g., file identifier) and its location (e.g., node address). A distributed hash table (DHT) for routing is usually implemented in this type of network. CAN, Chord, Pastry, and Tapestry are the first four DHTs introduced in academia [99]. Kademlia [83] is another DHT algorithm, which has been used in Overnet [11], eMule [5] and BitTorrent [2].

2.2 Botnets

Botnet, networks of compromised computers running malicious software and controlled by attackers, is one of the most significant threats to Internet. Therefore, botnet research is very active in the field of network security.

There have been some systematic studies on general botnets. Barfor and Yegneswaran [32] studied and compared four widely-used IRC-based botnets from seven aspects: botnet control mechanisms, host control mechanisms, propagation mechanisms, exploits, delivery mechanisms, obfuscation and deception mechanisms. Considering aspects such as attacking behavior, C&C model, rally mechanism, communication protocol, evasion technique and other observable activities, Trend Micro [85] proposed a taxonomy of botnet threads. In [46] Dagon *et al.*, also presented a taxonomy for botnets but from a different perspective. Their taxonomy focuses on the botnet structure and utility. And in 2008, a botnet research survey [131] done by Zhu *et al.*, classified recent research work on botnets into three categories: bot anatomy, wide-area measurement study and botnet modeling and future botnet prediction. Recently, Bailey *et al.*, presented another survey, which provided an overview of current

botnets, discussed how different types of networks can affect the effectiveness of botnet detection mechanism, and talked about various detection techniques that have been proposed [31].What differs our work from theirs is that we focus on newly appeared P2P botnets, and try to understand P2P botnets along four dimensions: P2P botnet construction, C&C mechanisms, measurements and defenses.

As we discussed earlier in Chapter 1, P2P botnets is a new form of botnets. They have appeared in recent years and obtained people's attention. In [57] Grizzard *et al.*, conducted a case study on Trojan.Peacomm botnet. Later on, Holz *et al.*, adapted tracking technique used to mitigate IRC-based botnets and extended it to analyze Storm worm botnets [64]. Trojan.Peacomm botnet and Stormnet are two typical P2P botnets. Although bots in these two botnets are infected by two different malware, Trojan.Peacomm and Storm worm respectively, both of their C&C mechanisms are based on Kademlia [83], which is a DHT routing protocol designed for decentralized P2P networks. And a botnet protocol which is also based on Kademlia was proposed by Starnberger *et al.* [107]. Moreover, to be well prepared for the future, some other botnets whose architecture is similar to P2P architecture, such as super botnet [121] and the Loosely Coupled P2P botnet (lcbot) [39], have been presented as well.

Researchers are also trying to model P2P botnet propagation. In recent work [100], Ruitenbeek and Sanders presented a stochastic model of the creation of a P2P botnet. In [47], Dagon *et al.*, proposed a diurnal propagation model for computer online/offline behaviors and showed that regional bias in infection will affect the overall growth of the botnet. [98] formulated an analytical model that emulates the mechanics of a decentralized Gnutella type of peer network and studied the spread of malware on such networks. Both [129] and [118] presented an analytical propagation model of P2P worms, but the former targets topological scan based P2P worms, while the latter targets passive scan based P2P worms.

Meanwhile, there has been an enormous amount of work on botnet detection and mitigation. Wurzinger *et al.* presented an approach to automatically generate models for botnet detection [127]. Their models are generated based on the fact that every bot responds to the

botmaster in a specific way. Researchers try to distinguish bot behavior from human behavior, in order to detect botnets. For example, in [81], malicious channels created by bots are differentiated from normal traffic generated by human beings; and in [60], hypothesis testing is used to separate botnet C&C dialogs from human-human conversations. Pattern recognition approaches and clustering algorithms are widely used for botnet detection. Chang and Daniels proposed a node behavior profiling approach to capture the node behavior clusters in a network for botnet C&C communication detection [38]. And a Bayesian approach for detecting bots based on the similarity of their DNS traffic to that of known bots is presented in [119]. In addition, Gu et al., proposed three botnet detection systems: BotMiner [58] – a botnet detection framework by performing cross cluster correlation on captured communication and malicious traffic, BotSniffer [61] – a system that can identify botnet C&C channels in a local area network based on the observation that bots within the same botnet will demonstrate spatial-temporal correlation and similarity and BotHunter [59] – a bot detection system using IDS-Driven Dialog Correlation according to defined bot infection dialog model.

Furthermore, botnet infiltration and monitoring is also an very active topic in botnet research community. Li et al. [78], monitored botnets probing activities and addressed the problems like botnet's scanning strategies and attack target selection policies. In [70], Kanich et al., pointed out a number of challenges that arise in using crawling to measure the size, topology, and dynamism of distributed botnets. People infiltrate specific botnets, such as MegaD botnet [42], Torpig bot [113] and [88], in order to understand their architectures, communication protocols, behaviors, etc. In addition, infiltration and monitoring can be very helpful for fighting against malicious activities. In [90,75,76], the data collected through infiltrating and monitoring botnets are used for spam detection and analysis.

2.3 Honeypots

A honeypot is a special constructed computer or network trap designed to attract and detect malicious attacks [54]. Two or more honeypots on a network form a honeynet.

Honeypots and honeynets are effective detection and defense techniques, and hence there has been much recent research in this area. Provos [93] presented "honeyd," a honeypot software package that makes large-scale honeynet monitoring possible. Dagon *et al.* [45] presented the "HoneyStat" system to use coordinated honeypots to detect worm infections in local networks. Jiang and Xu [67] presented a virtual honeynet system that has a distributed presence and centralized operation. Bailey *et al.* [30] presented a globally distributed, hybrid, honeypot-based monitoring architecture which deploys low-interaction honeypots as the frontend content filters and high-interaction honeypots to capture detailed attack traffic. Vrable *et al.* [122] presented several effective methods to design large-scale honeynet systems capable of obtaining high-fidelity attack data, which they called "Potemkin". Tang and Chen [117] presented a novel "double-honeypot" detection system to effectively detect Internet worm attacks. Anagnostakis *et al.* [26] presented a way to use a "shadow honeypot" to conduct real-time host-based attack detection and defense.

Furthermore, in the field of fighting against botnets, McCarty [84] discussed how to use a honeynet to monitor botnets. There are two techniques to monitor botnet activities. The first technique is to allow honeypots or honeynets to be compromised and join a botnet [43,55,29]. Behaving as normal "bots" in the botnet, these honeypot spies provide valuable information of the monitored botnet activities. With the help from Dynamic DNS service providers, the second technique is to hijack C&C server in botnets to monitor the command and control communications in botnets [47]. This was accomplished by redirecting the servers' DNS mapping to a botnet monitor.

There has been some research in discovering and concealing honeypots. Provos [93] discussed how to vividly simulate the routing topology and services of a virtual network by

tailoring honeyd's response. GenII honeynets [92] allow a limited number of packets to be sent out from an infected honeynet. From the botmaster's perspective, some hardware or software specific means have always been available to detect infected honeypots (e.g. by detecting VMware [4] or another emulated virtual environment [102, 44], or by detecting the honeypot program's faulty responses [21].) However, there has been no systematic research on honeypot detection based on a general methodology.

Krawetz [73] introduced the commercial anti-honeypot spamming tool, "Send-Safe's Honeypot Hunter". On a spammer's computer, the tool is used to detect honeypot open proxies by testing whether the remote open proxy can send email back to the spammer. This anti-honeypot tool uses the similar approach presented in this paper. It can be treated as a special realization of the methodology presented here, but it is only effective for detecting open proxy honeypots.

Bethencourt *et al.* [34] presented a method for attackers to use intelligent probings to detect the location of Internet security sensors (including honeypots) based on their public report statistics. In this dissertation, we present a general honeypot detection approach that does not require a honeypot to publish its monitored statistics.

CHAPTER 3: PEER-TO-PEER BOTNET DISSECTION

3.1 Introduction

The life cycle of botnets is composed of three stages. Stage one - recruiting members, a botmaster needs to compromise many computers in the Internet, so that he/she can control them remotely. All kinds of malicious software, such as worms, Trojan houses, viruses, instance message (IM) malware, can be used to accomplish this task. Stage two - forming the botnet, bots need to find a way to connect to each other and form a botnet. Stage three - standing by for instructions, after the botnet is built up, all bots are ready to communicate with their botmaster for further instructions, such as launching an attack or performing an update. There is no exception for P2P botnets. In this chapter, we will study each stage of P2P botnet's life cycle in detail. Our work includes the following points:

- Bootstrap procedure as a possible single point of failure during botnet construction is well studied;
- Two command and control mechanisms are presented and explained;
- We discuss the similarities that P2P botnets share with traditional centralized botnets and unique features that P2P botnets have;
- We categorize P2P botnets into three classes, and discuss their features and limitations.

As we mentioned in Chapter 2, there have been some systematic studies on general botnets. What differs our work from theirs is that we specifically focus on newly appeared P2P botnets, and try to understand P2P botnets along multiple dimensions: infection vectors, bot candidate selection, bootstrap procedure, network structure, C&C mechanisms and communication protocols.

3.2 Stage One: Recruiting Bot Members

P2P networks are gaining popularity of distributed applications, such as file-sharing, web caching, network storage [24]. In these content trading P2P networks, without a centralized authority it is not easy to guarantee that the file exchanged is not malicious. For this reason, these networks become the ideal venue for malicious software to spread. It is straightforward for attackers to target vulnerable hosts in existing P2P networks as bot candidates and build their zombie army. Many P2P malware have been reported, such as W32/Gnuman [19], VBS.Gnutella [18] and SdDrop [14]. They can be used to compromise a host and make it become a bot.

However, in this way, the scale of a botnet will be limited by the size of the existing P2P network, and the network will be the only propagation media. On the contrary, P2P botnets we have witnessed recently [57, 64, 114] do not confine themselves to existing P2P networks. They have shown that it is more flexible and practical if bot members are recruited from the entire Internet through all possible spread mediums like emails, instant messages and file exchange.

3.3 Stage Two: Forming the Botnet

Upon infection, the next important thing is to let newly compromised computers join the botnet network and connect to other bots. Otherwise, they are just isolated individual computers without much use for botmasters.

Now for the convenience of further discussion, we first introduce three terms: "*parasite*", "*leeching*" and "*bot-only*" P2P botnets. Each of them represents a class of P2P botnets. In a parasite P2P botnet, all the bots are selected from vulnerable hosts within an existing P2P network, and it uses this available P2P network for command and control. A leeching P2P botnet refers to one whose members join an existing P2P network and depend on this P2P network for C&C communication, but the bots could be vulnerable hosts that were

either inside or outside of the existing P2P network. For example, the early version of Storm botnet belongs to this class of botnet. A bot-only P2P botnet builds its own network, in which all the members are bots, such as Stormnet and Nugache.

If all the bots are selected from an existing P2P network, it is not necessary to perform any further action to form the botnet, because bots can find and communicate with each other by simply using current P2P protocol. In other words, for a parasite P2P botnet, up to this point, the botnet construction is done and the botnet is ready to be operated by its botmaster.

However, if a random host on the Internet is compromised, it has to know how to find and join the botnet. As we know current P2P file-sharing networks provide the following two general ways for new peers to join a network:

(i) An initial peer list is hard-coded in each P2P client. When a new peer is up, it will try to contact peers in that initial list to update its neighboring peer information.

(ii) There is a shared web cache, such as Gnutella web cache [48], stored at some place on the Internet, and the location of the cache is put in the client code. Thus new peers can refresh its neighboring peer list by going to the web cache and fetching the latest updates.

This initial procedure of finding and joining a P2P network is usually called "bootstrap" procedure. It can be directly adopted for P2P botnet construction. Either a predetermined list of peers or the locations of predetermined web caches need to be hard-coded in the bot code. Then a newly infected host knows which peer to contact or at least where to find candidates of neighboring peers it will contact later.

For instance, Trojan.Peacomm [57] is a piece of malware to create a P2P botnet which uses the P2P protocol implemented on Overnet for C&C communication. A list of Overnet nodes that are likely to be online is hard-coded into the bot's installation binary. When a victim is compromised and runs a Trojan.Peacomm, it will try to contact peers in this list

to bootstrap onto the Overnet network. Another P2P botnet, Stormnet [64], uses a similar bootstrap mechanism: the information about other peers with which a new bot member communicates after the installation phase, is encoded in a configuration file that is also stored on the victim machine by Storm worm binary.

However, bootstrap is a vulnerable procedure and it could become a single point of failure for botnet construction. If the initial list of peers or web cache is obtained by defenders, they may be able to prevent botnet from growing simply by shutting down those bootstrap peers or web caches. From this perspective, we can see that although a parasite P2P botnet has limitations on bot candidate selection, it does not have the bootstrap vulnerability by choosing bot candidates from an existing P2P network.

To overcome this vulnerability, botnet attackers may think of other ways to avoid introducing bootstrap procedure in P2P botnet construction. For example, in the hybrid P2P botnet we will present in Chapter 4, instead of relying on hard-coded list, peer list exchange and mixing is introduced, in order to avoid bootstrap procedure. In [121] a similar procedure was presented to construct a super botnet instead of bootstrapping.

3.4 Stage Three: Standing By for Instructions

Once a botnet is built up, all the bots are standing by for instructions from their botmaster to perform illicit activities or updates. Therefore C&C mechanism is very important and is the major part of a botnet design. It directly determines the communication topology of a botnet, and hence affects the robustness of a botnet against network/computer failures, or security monitoring and defenses.

The C&C mechanisms can be categorized as either *pull* or *push* mechanism. Pull mechanism, i.e., "command publishing/subscribing", refers to the manner that bots retrieve commands actively from a place where botmasters publish commands. On the contrary, push

mechanism, i.e., "command forwarding", means bots passively wait for commands to come and will forward received commands to others.

For centralized botnets, pull mechanism is commonly used. Take HTTP-based botnets as an example, a botmaster publishes commands on a web page, and bots periodically visit this web page via HTTP to check for any command updates. In the following, we will discuss how pull and push C&C mechanisms can be applied in P2P botnets.

3.4.1 Leverage Existing Peer-to-Peer Protocols

As we discussed above, both parasite and leeching P2P botnets depend on existing P2P networks. Thus it is natural to leverage the existing P2P protocols used by the host P2P networks for C&C communication. Besides, these protocols have been tested in P2P file-sharing applications for a long time, so they tend to be less error-prone than newly designed ones, and have nice properties to improve performance of P2P systems and mitigate network problems, such as link failure or churn. The following discussion is based on parasite and leeching P2P botnets, but bot-only botnet can adopt these protocols as well.

In P2P file-sharing systems, file index which is used by peers to locate the desired content, may be centralized (e.g., Napster), distributed over a fraction of the file-sharing nodes (e.g., Gnutella), or distributed over all or a large fraction of the nodes (e.g., Overnet). A peer can send out query message for the file it is searching for, and the message will be passed around according to the routing algorithm implemented in the system. The search will terminate when query hits are returned or the query message expires.

Botmasters can easily adopt the above procedure to disseminate commands in pull style. They can insert records associated with some predefined file titles or hash values into the index, but rather than putting the content location information, botnet commands are attached. In order to get commands issued by botmasters, bots periodically initiate queries for those files or hashes, and nodes who preserve the corresponding records will return query

hits with commands encoded. In other words, bots subscribe the content, i.e., commands, published by botmaster.

The early version of Storm botnet [64] is a good example to show how a P2P botnet could leverage an existing P2P network or implement an existing P2P protocol for the pull-style command and control, although it uses the Overnet P2P network to pass the locations of its C&C servers instead of botmaster's commands. In Storm botnet, every day there are 32 keys queried by bots to retrieve important information. These 32 keys are calculated by a built-in algorithm, which takes the current date and a random number from [0-31] as input. Therefore, when issuing a command, the botmaster needs to publish it under 32 different keys. Trojan.Peacomm botnet [57] employs the similar design.

Compared to pull mechanism, implementation of push mechanism on existing P2P protocols is more complex. There are two major design issues:

- Which peers should a bot forward a command to?
- How to forward commands: using in-band (normal P2P traffic) or out-of-band messages (non-P2P traffic)?

To address the first issue, the simplest way is to let a bot use its current neighboring peers as targets. But the problem of this approach is that command distribution may be slow or sometimes disrupted, because 1) some bots have a small number of neighbors, or 2) some peers in a bot's neighbor list are bot members in the case of parasite or leeching P2P botnets. One solution to this problem is that letting bots claim they have certain popular files available which are predefined, and forwarding commands to peers appearing in the search results for those files. Thus the chance of commands hitting an actual bot is increased. These predefined popular files behave as the watchwords for the botnet, but could give defenders a clue to identify bots.

For the second issue, whether using in-band or out-of-band message to forward a command depends on what the peers in the target list are. If a bot targets its neighboring peers,

in-band message is a good choice. A bot could encode a command in a query message, which can only be interpreted by bots, send it to all its neighbors, and rely on them to continue passing on the command in the botnet. This scheme is easy to implement and hard for defenders to detect, because there is no difference between command forwarding traffic and normal P2P traffic. On the other hand, if the target list is generated in other ways, like using peers in returned search results discussed above, bots have to contact those peers using out-of-band message. Obviously out-of-band traffic are easier to detect, and hence, can disclose the identities of bots who initiate such traffic.

The above discussion mainly focused on unstructured P2P networks, where query messages are flooded to the network. In structured P2P networks (e.g., Overnet), a query message is routed to the nodes whose node IDs are closer to the queried hash key, which means queries for the same hash key are always forwarded by the same set of nodes. Therefore, to let more bots receive a command, the command should be associated with different hash keys, such that it can be sent to different parts of the network.

3.4.2 Design a New Peer-to-Peer Communication Protocol

It is convenient to adopt existing P2P protocols for P2P botnet C&C communication, however, the inherited drawbacks may limit botnet design and performance. A botnet can be more flexible if it uses a new protocol designed by its botmaster.

The advanced hybrid P2P botnet [124] presented in Chapter 4 and the super botnet [121] are two newly designed P2P botnets, whose C&C communication are not dependent on existing P2P protocols. Both of them implements push and pull C&C mechanisms. In a hybrid P2P botnet, when a bot receives a command, it forwards the command to all the peers in the list (push), and those who cannot accept connection from others periodically contacts other bots in the list and try to retrieve new commands (pull). A super botnet is composed of a number of small centralized botnets. Commands are pushed from one small

botnet to another, and within a small centralized botnet, bots pull the command from their C&C servers. Furthermore, the hybrid P2P botnet is able to effectively avoid bootstrap procedure, which is required by most of the existing P2P protocols, by 1) passing a peer list from one bot to a host that is infected by this bot, and 2) exchanging peer lists when two bots communicate.

The drawback of designing a new protocol for P2P botnet communication is that the new protocol has never been tested before. When a botnet using this protocol is deployed, the network may not be as stable and robust as expected due to complex network conditions and defenses.

3.4.3 Discussion

Several features can be extracted to represent a P2P botnet during its life cycle: infection vectors, bot candidates, bootstrap procedure, members in the network, communication protocols and C&C styles. Parasite, leeching and bot-only P2P botnets, they share common features but differ in others, which have been summarized in Table 3.1. It is shown that parasite P2P botnets are less flexible except that no bootstrap is required. This is an advantage of the parasite over the other two classes. During bootstrap, botnets are vulnerable and the propagation could be stopped if bootstrap information is compromised by defenders. Leeching and bot-only P2P botnets are more like each other, but the former ones are more stealthy, because they resides in existing P2P networks, and bot members are mixed with legitimate nodes and not easy to be detected.

3.5 Summary

Like all the botnets, P2P botnets also have to go through three stages — bot selection, network construction and command and control communication, to complete the entire process of botnet construction and operation. But P2P botnets have their own features which

Table 3.1: Comparison among three types of P2P botnets

Features	**Parasite**	**Leeching**	**Bot-only**
Infection vectors	P2P malware	Any kind of malware	Any kind of malware
Bot candidates	Vulnerable hosts in P2P networks	Vulnerable hosts in the Internet	Vulnerable hosts in the Internet
Bootstrap procedure	None	Required	Optional
Members in the network	legitimate peers and bots	legitimate peers and bots	Only bots
Communication protocols	Existing P2P protocols	Existing P2P protocols	Self-designed or existing P2P protocols
C&C styles	Pull or push	Pull or push	Pull or push

make them different from centralized botnets. In this chapter, we carefully studied each stage of the life cycle of P2P botnets. Besides traditional mediums which is used to spread malware, such as email, websites, etc, the existing P2P file sharing networks are good venue for P2P botnet to recruit. Many mature techniques developed for P2P network construction and communication can be leveraged by botmasters to build their P2P botnets as well. Of course, new approach for network construction and new protocols for C&C communication can be and some have already been designed for P2P botnets, in order to remedy the limitations of P2P networks, such as the vulnerable bootstrap procedure.

CHAPTER 4: AN ADVANCED HYBRID PEER-TO-PEER BOTNET

4.1 Introduction

As botnet-based attacks become popular and dangerous, security researchers have studied how to detect, monitor, and defend against them [55, 43, 69, 47, 36, 97]. Most of the current research has focused upon the centralized botnets that have appeared in the past, especially IRC-based botnets. It is necessary to conduct such research in order to deal with the threat we are facing today. However, it is equally important to conduct research on advanced botnet designs that could be developed by attackers in the near future. Otherwise, we will remain susceptible to the next generation of internet malware attacks.

From a botmaster's perspective, the C&C servers are the fundamental weak points in current botnet architectures. First, a botmaster will lose control of the botnet once the limited number of C&C servers are shut down by defenders. Second, defenders could easily obtain the identities (e.g., IP addresses) of all C&C servers based on their service traffic to a large number of bots [36], or simply from one single captured bot (which contains the list of C&C servers). Third, an entire botnet may be exposed once a C&C server in the botnet is hijacked or captured by defenders [47]. As network security practitioners put more resources and effort into defending against botnet attacks, hackers will develop and deploy the next generation of botnets with a different control architecture, such as peer-to-peer structured botnets.

In this chapter, we first start with analyzing the weakness of current P2P botnets, then based upon our systematically study on P2P botnet in Chapter 3, we address the issues of P2P botnets, and propose an advanced hybrid P2P botnet, which has many features to make it robust and resilient to defenses. Our research work includes the following major aspects of the proposed advanced hybrid P2P botnets:

Design details

- We give comprehensive descriptions on the botnet architecture, encrypted command and control communication and botnet monitoring and management.

- For network construction, besides the basic procedure we present, peer list updating procedure is introduced in order to balance the connectivity of the botnet topology.

Robustness study

- We introduce two robustness metrics.

- Simulation-based experiments are conducted to evaluate the robustness of proposed P2P botnet in terms of those two metrics.

- We present mathematical robustness analysis for proposed botnet as well, and use simulation-based experiment to verify the accuracy of our analysis.

Defenses

- We discuss defenses against proposed botnet from three directions: annihilation, honeypot-based botnet monitoring, and botnet detection and monitoring without honeypots.

- We present mathematical analysis on botnet monitoring with and without honeypots.

- We also perform simulation-based experiments to show the effectiveness of botnet monitoring based on honeypot techniques.

4.1.1 Current Peer-to-Peer Botnets and Their Weaknesses

As we previously discussed, considering the weaknesses in the architecture of centralized botnets, it is a natural strategy for botmasters to design a P2P control mechanism

into their botnets. In the last several years, botnets such as Slapper [28], Sinit [15], Phatbot [12] and Nugache [77] have implemented different kinds of P2P control architectures. They have shown several advanced designs. For example, some of them have removed the "bootstrap" process used in common P2P protocols[1]. Sinit uses public key cryptography for update authentication [15]. Nugache attempts to thwart detection by implementing an encrypted/obsfucated control channel [77].

Nevertheless, simply migrating available P2P protocols will not generate a sound botnet, and the P2P designs used by several botnets in the past are not mature and have many weaknesses. To remove bootstrap procedure, a Sinit bot uses random probing to find other Sinit bots to communicate with. This results in poor connectivity for the constructed botnet and easy detection due to the extensive probing traffic [15]. Phatbot utilizes Gnutella cache servers for its bootstrap process. This botnet can be easily shut down if security community set up filter on those Gnutella cache servers, or block any traffic to and from those cache servers. In addition, its underlying WASTE peer-to-peer protocol is not scalable across a large network [12]. Nugache's weakness lies in its reliance on a seed list of 22 IP addresses during its bootstrap process [77]. Slapper fails to implement encryption and command authentication enabling it to be easily hijacked by others. In addition, its list of known bots contains all (or almost all) members of the botnet. Thus, one single captured bot would expose the entire botnet to defenders [28]. Furthermore, its complicated communication mechanism generates a large amount of traffic, rendering it susceptible to monitoring via network flow analysis.

Some other available distributed systems include "censorship-resistant" system and "anonymous" P2P system. However, their design goal is different from a botnet. For example, these

[1]Many peer-to-peer networks have a central server or a seed list of peers who can be contacted in order for a new peer joining the network. This bootstrap process is not a major problem for normal P2P networks, but it poses a single point of failure for a P2P botnet.

distributed systems try to hide the source node of a message within a crowd of nodes. However, they do not bother to hide the identities of this crowd. On the other hand, a botnet needs to try its best to hide IP addresses of all bots in it.

4.1.2 Proposed Hybrid Peer-to-Peer Botnet

Considering the problems encountered by centralized botnets and previous P2P botnets, the design of an advanced botnet, from our understanding, should consider the following practical challenges faced by botmasters: (1). How to generate a robust botnet capable of maintaining control of its remaining bots even after a substantial portion of the botnet population has been removed by defenders? (2). How to prevent significant exposure of the network topology when some bots are captured by defenders? (3). How to easily monitor and obtain the complete information of a botnet by its botmaster? (4). How to prevent (or make it harder for) defenders from detecting bots via their communication traffic patterns? In addition, the design should also consider many network related issues such as dynamic or private IP addresses and the diurnal online/offline property of bots [47].

By considering all the challenges listed above, we present our research on the possible design of an advanced hybrid P2P botnet. The proposed hybrid P2P botnet has the following features:

- The botnet requires no bootstrap procedure.
- The botnet communicates via the peer list contained in each bot. However, unlike Slapper [28], each bot has a fixed and limited size peer list and does not reveal its peer list to other bots. In this way, when a bot is captured by defenders, only the limited number of bots in its peer list are exposed.
- A botmaster could easily monitor the entire botnet by issuing a *report* command. This command instructs all (or partial) bots to report to a compromised machine (which is called a *sensor host*) that is controlled by the botmaster. The IP address of the

sensor host, which is specified in the report command, will change every time a report command is issued to prevent defenders from capturing or blocking the sensor host beforehand.

- After collecting information about the botnet through the above report command, a botmaster, if he/she thinks necessary, could issue an *update* command to actively let all bots contact a sensor host to update their peer lists. This effectively updates the botnet topology such that it has a balanced and robust connectivity, and/or reconnects a broken botnet.
- Only bots with static global IP addresses that are accessible from the Internet are candidates for being in peer lists (they are called *servent bots* according to P2P terminologies since they behave with both client and server features). This design ensures that the peer list in each bot has a long lifetime.
- Each servent bot listens on a self-determined service port for incoming connections from other bots and uses a self-generated symmetric encryption key for incoming traffic. This individualized encryption and individualized service port design makes it very hard for the botnet to be detected through network flow analysis of the botnet communication traffic.

4.2 Proposed Hybrid Peer-to-Peer Botnet Architecture

4.2.1 Two Classes of Bots

The bots in the proposed P2P botnet are classified into two groups. The first group contains bots that have static, non-private IP addresses and are accessible from the global Internet. Bots in the first group are called *servent bots* since they behave as both clients and servers[2]. The second group contains the remaining bots, including: (1). Bots with

[2]In a traditional peer-to-peer file sharing system, all hosts behave both as clients and servers and are called "servents" [82].

dynamically allocated IP addresses; (2). Bots with private IP addresses; (3). Bots behind firewalls such that they cannot be connected from the global Internet. The second group of bots are called *client bots* since they will not accept incoming connections.

Only servent bots are candidates in peer lists. All bots, including both client bots and servent bots, actively contact the servent bots in their peer lists to retrieve commands. Because servent bots normally do not change their IP addresses, this design increases the network stability of a botnet. This bot classification will become more important in the future as a larger proportion of computers will sit behind firewall, or use DHCP or private IP addresses due to shortage of IP space.

A bot could easily determine the type of IP address used by its host machine. For example, on a Windows machine, a bot could run the command "`ipconfig /all`". Not all bots with static global IP addresses are qualified to be servent bots—some of them may stay behind firewall, inaccessible from the global Internet. A botmaster could rely on the collaboration between bots to determine such bots. For example, a bot runs its server program and requests the servent bots in its peer list to initiate connections to its service port. If the bot could receive such test connections, it labels itself as a servent bot. Otherwise, it labels itself as a client bot.

4.2.2 Botnet Command and Control Architecture

Fig. 4.1 illustrates the C&C architecture of the proposed botnet. The illustrative botnet shown in this figure has 5 servent bots and 3 client bots. The peer list size is 2 (i.e. each bot's peer list contains the IP addresses of 2 servent bots). An arrow from bot A to bot B represents bot A initiating a connection to bot B.

A botmaster injects his or her commands through any bot(s) in the botnet. Both client and servent bots actively and periodically connect to the servent bots in their peer lists in order to retrieve commands issued by their botmaster. When a bot receives a new command

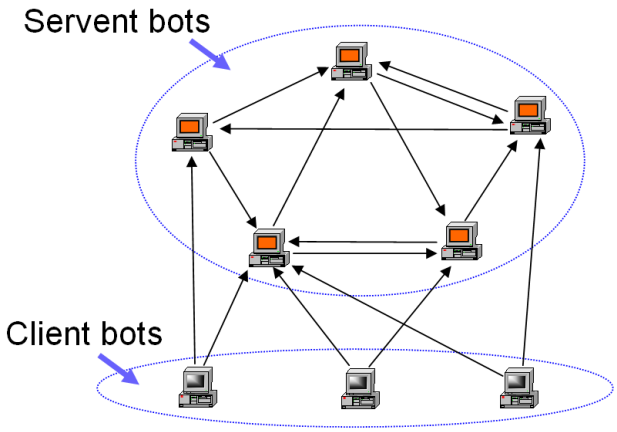


Figure 4.1: Command and control architecture of proposed hybrid P2P botnet

that it has never seen before (e.g., each command has a unique ID), it immediately forwards the command to all servent bots in its peer list.

This description of command communication means that, in terms of command forwarding, the proposed botnet has an undirected graph topology. A botmaster's command could pass via the links shown in Fig. 4.1 in both directions. If the size of the botnet peer list is denoted by M, then this design makes sure that each bot has at least M venues to receive commands.

4.2.3 Relationship Between Centralized Botnets and the Proposed Botnet

Compared to a centralized botnet (see Fig. 1.1(a)), it is easy to see that the proposed hybrid P2P botnet shown in Fig. 4.1 is actually an extension of a centralized botnet. The hybrid P2P botnet is equivalent to a centralized botnet where servent bots take the role of C&C servers: the number of C&C servers (servent bots) is greatly enlarged, and they interconnect with each other. Indeed, the large number of servent bots is the primary reason why the proposed hybrid P2P botnet is very hard to be shut down. We will explain these properties in detail later in Section 4.5 and Section 4.6.

4.3 Botnet Command and Control

The essential component of a botnet is its C&C communication. Compared to a centralized botnet, the proposed botnet has a more robust and complex communication architecture. The major design challenge is to generate a botnet that is difficult to be shut down, or monitored by defenders or other attackers.

4.3.1 Command Authentication

Compared with a centralized botnet, because bots in the proposed botnet do not receive commands from predefined places, it is especially important to implement a strong command authentication. A standard public-key authentication would be sufficient. A botmaster generates a pair of public/private keys, $\langle K^+, K^- \rangle$, and hard codes the public key K^+ into the bot program before releasing and building the botnet. There is no need for key distribution because the public key is hard-coded in bot program. Later, the command messages sent from the botmaster could be digitally signed by the private key K^- to ensure their authentication and integrity.

This public-key based authentication could also be readily deployed in current centralized botnets. So botnet hijacking is not a major issue.

4.3.2 Individualized Encryption Key

In the proposed botnet, each servent bot i randomly generates its symmetric encryption key K_i. Suppose the peer list on bot A is denoted by L_A. It will not only contain the IP addresses of M servent bots, but also the symmetric keys used by these servent bots. Thus, the peer list on bot A is:

$$L_A = \{(\mathrm{IP}_{i_1}, K_{i_1}), (\mathrm{IP}_{i_2}, K_{i_2}), \cdots (\mathrm{IP}_{i_M}, K_{i_M})\} \qquad \text{(Eq. 4.1)}$$

where $(\text{IP}_{i_j}, K_{i_j})$ are the IP address and symmetric key used by servent bot i_j. With such a peer list design, each servent bot uses its own symmetric key for incoming connections from any other bot. This is applicable because if bot B connects to a servent bot A, bot B must have (IP_A, K_A) in its peer list.

This *individualized* encryption guarantees that if defenders capture one bot, they only obtain keys used by M servent bots in the captured bot's peer list. Thus the encryption among the remaining botnet will not be compromised.

4.3.3 Individualized Service Port

The peer-list based architecture also enables the proposed botnet to disperse its communication traffic in terms of service port. Since a servent bot needs to accept connections from other bots, it must run a server process listening on a service port. The service port number on servent bot i, denoted by P_i, could be randomly picked by the bot. Considering this, a peer list needs to contain the service port information as well. For example, the peer list on bot A is:

$$L_A = \{(\text{IP}_{i_1}, K_{i_1}, P_{i_1}), \cdots, (\text{IP}_{i_M}, K_{i_M}, P_{i_M})\} \quad \text{(Eq. 4.2)}$$

The individualized service port design has two benefits for botmasters:

- **Dispersed network traffic:** Since service port is a critical parameter in classifying network traffic, this individualized port design makes it extremely hard for defenders to detect a botnet based on monitored network traffic. When combined with the individualized encryption design, a P2P botnet has a strong resistance against most (if not all) network traffic flow based detection systems, such as the ones introduced in [40, 115].

- **Secret backdoor:** The individualized port design also ensures that servent bots in a P2P botnet keep their backdoors "secret". Otherwise, defenders could scan the

specific port used by a botnet to detect potential servent bots, or monitor network traffic targeting this service port to facilitate their botnet detection.

A randomly-generated service port may not always be good for botnets since network traffic going to a rarely used port is abnormal. To overcome this, a botmaster can specify a set of service ports for each bot to choose, preferably choosing from those standard encrypted ports such as port 22 (SSH), port 443 (HTTPS), or port 993 (IMAPS). Furthermore, a sophisticated botmaster could even program bot code to mimic the protocol format of the service port as what "honeyd" [93] does.

4.4 Botnet Monitoring by Its Botmaster

Another major challenge in botnet design is making sure that a botnet is difficult to be monitored by defenders, but at the same time, easily monitored by its botmaster. With detailed botnet information, a botmaster could (1). conduct attacks more effectively according to the bot population, distribution, bandwidth, on/off status, IP address types, etc; (2). keep tighter control over the botnet when facing various counterattacks from defenders. In this section, we present a simple but effective way for botmasters to monitor their botnets whenever they want, and at the same time, resist being monitored by others.

4.4.1 Monitoring via a Dynamically Changeable Sensor

To monitor the proposed hybrid P2P botnet, a botmaster issues a special command, called a *report* command, to the botnet thereby instructing every bot to send its information to a specified machine that is compromised and controlled by the botmaster. This data collection machine is called a *sensor*.

The IP address (or domain name) of the centralized sensor host is specified in the report command. Every round of report command issued by a botmaster could potentially utilize a different sensor host. This would prevent defenders from knowing the identity of the sensor

host before seeing the actual report command. After a report command has been sent out by a botmaster, it is possible that defenders could quickly know the identity of the sensor host (e.g., through honeypot joining the botnet [55, 43]), and then either shut it down or monitor the sensor host. To deal with this threat, a botmaster may implement any of the following procedures:

- Use a popular Internet service, such as HTTP or Email, for report to a sensor. The sensor is chosen such that it normally provides such a service to avoid exhibiting abnormal network traffic.
- Use several sensor machines instead of a single sensor.
- Select sensor hosts that are harder to be shut down or monitored, for example, compromised machines in other countries with minimum Internet security and International collaboration.
- Manually verify the selected sensor machines are not honeypots (see further discussion in Section 4.8).
- Wipe out the hard drive on a sensor host immediately after retrieving the report data.
- Specify expiration time in report command to prevent any bot exposing itself after that time.
- Issue another command to the botnet to cancel the previous report command once the botmaster knows that the sensor host has been captured by defenders.

If a botmaster simply wants to know the current size of a botnet, a probabilistic report would be preferred: each bot uses a small probability p specified in a report command to decide whether to report. Then the botnet has roughly X/p bots if X bots report. Such a probabilistic report could minimize the telltale traffic to the report sensor.

Each bot could use its hard-coded public key K^+ to ensure the confidentiality of its report data. In addition, a botmaster could use several compromised machines as stepping

stones in retrieving the data on sensors. These are standard practices so we will not explain more.

4.4.2 Additional Monitoring Information

A botmaster not only wants to know a botnet size and topology, he/she may also want to know other information in order to conduct efficient attacks.

4.4.2.1 IP Address Type

Internet computers are identified by their IP addresses. But the widely-deployed "Dynamic Host Configuration Protocol" (DHCP) and "Network Address Translation" (NAT) have made IP-based identification difficult and error-prone. A botmaster may implement an ID-based identification (such as [35]) to track bots in a botnet: when compromising a computer, the bot program on the computer randomly generates its unique ID. When bots send their report to a sensor host, they report their IDs as well.

This ID-based identification facilitates the measurement of NAT encountered by a botnet. Based on a round of report sent from bots, a botmaster is able to know whether several bots are behind a single NAT — these bots have different IDs but send reports from a single source IP address.

This ID-based identification eliminates the measurement trouble caused by DHCP addresses. A more useful measurement is to know how frequently DHCP-based bots in a botnet change their IP addresses. For this purpose, each bot with DHCP address keeps recording when its IP address changes, and then report this information to its botmaster's sensor host.

This information is particularly useful for botmasters in sending email spam. As pointed out by [96], 80% of current spam is listed in some DNS blacklists (DNSBLs), which are used "to track IP addresses that originate spam, so that future emails sent from these IP addresses can be rejected." [97] This means that if a botnet sends out spam, many bots in the botnet

will lose their capability to send out spam again. To counterattack this defense, a botmaster may only use DHCP bots, which change their IP addresses, for example, at least once per day, to send out email spam. In this way, defenders need to blacklist a much larger number of IPs to effectively block spam, and it becomes much harder for defenders to determine a good timeout value for blocked IPs.

Because a botmaster knows how many bots satisfy this requirement based on botnet report, the botmaster can strike a good tradeoff between sending out enough spam and avoiding being blocked by DNSBLs.

The ID-based identification has another benefit: if a botmaster wants to let a selected set of bots to send out an attack, the botmaster may issue a command to the botnet with a list of these attack bots' IDs — a bot will launch attack if it finds its ID in this list. In this way, if the list is captured by defenders, it will not reveal the identities of these attack bots since only the botmaster knows the mapping between IP addresses and IDs.

4.4.2.2 Diurnal Dynamics

As pointed out in [47], the online population of every botnet has a clear "diurnal" dynamics due to many users shutting down their computers at night. In one time zone, the peak online population of a botnet could be as much as four times of the bottom level online population. The significance of diurnal dynamics is further verified by [56], which showed that only about 20% of computers are always online.

To maximize a botnet attack power, a botmaster may want to know the diurnal dynamics of the botnet. For example, a botmaster can launch a denial-of-service attack at the right time when the botnet online population reaches its peak level, or spread a new malware at the optimal release time to increase its propagation speed as introduced in [47].

The diurnal dynamics of each bot is not hard to obtain since a bot is in fact a spyware. For example, at the beginning of each hour, a running bot program appends the current

time to a data file, which is then reported to its botmaster. Based on such report data, a botmaster can derive the accurate diurnal dynamics of each bot.

4.5 Botnet Construction

4.5.1 Basic Construction Procedures

Botnet connectivity is solely determined by the peer list in each bot. A natural way to build peer lists is to construct them during propagation. To make sure that a constructed botnet is connected, the initial set of bots should contain some servent bots whose IP addresses are in the peer list on every initial bot. Suppose the size of peer list in each bot is configured to be M. As a bot program propagates, the peer list in each bot is constructed according to the following procedures:

- **New infection:** Bot A passes its peer list to a vulnerable host B when compromising it. if A is a servent bot, B adds A into its peer list (by randomly replacing one entry if its peer list is full). If A knows that B is a servent bot (A may not be aware of B's identity, for example, when B is compromised by an email virus sent from A), A adds B into its peer list in the same way.
- **Reinfection:** If reinfection is possible and bot A reinfects bot B, bot B will then replace R ($R \leq M - 1$) randomly-selected bots in its peer list with R bots from the peer list provided by A. Again, bot A and B will add each other into their respective peer lists if the other one is a servent bot as explained in the above "new infection" procedure.

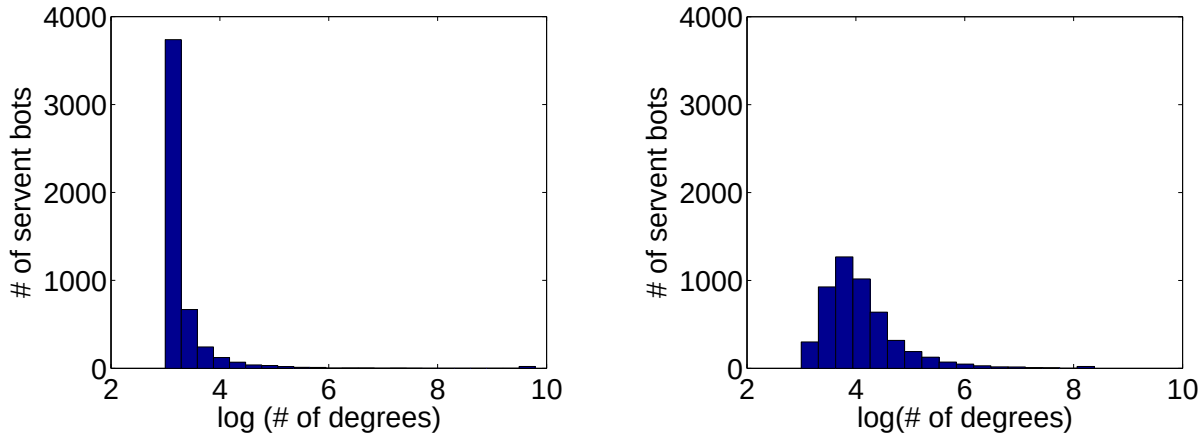
When reinfection happens frequently, the reinfection procedure can effectively interconnect different infection paths together, making a botnet evenly connected. In addition, this procedure makes it hard for defenders to infer the infection time order ("traceback") among bots based on captured peer lists.

In the reinfection procedure, a bot does not provide its peer list to those who reinfect it. This is important, because, if not, defenders could *recursively* infect (and monitor) all servent bots in a botnet based on a captured bot in their honeypot in the following way: Defenders use a firewall redirecting the outgoing infection attempts from captured bot A to reinfect the servent bots in A's peer list; then subsequently get the peer lists from these servent bots and reinfect servent bots in these peer lists in turn.

In order to study a constructed botnet topology and its robustness via simulations, we first need to determine simulation settings. First, Bhagwan *et al.* [35] studied P2P file sharing systems and observed that around 50% of computers changes their IP addresses within four to five days. So we expect the fraction of bots with dynamic addresses is around the similar range. In addition, some other bots are behind firewalls or NAT boxes so that they cannot accept Internet connections. We cannot find a good source specifying this statistics, so in this dissertation we assume that 25% of bots are servent bots.

Second, as pointed out in [22, 23], botnets in recent years have dropped their sizes to an average of 20,000, even though the potential vulnerable population is much larger. Thus we assume a botnet has a potential vulnerable population of 500,000, but stops growing after it reaches the size of 20,000. In addition, we assume that the peer list has a size of $M = 20$ and that there are 21 initial servent hosts to start the spread of the botnet. In this way, the peer list on every bot is always full.

Because scanning and vulnerability exploit is the dominant infection mechanism used by current botnets, in this dissertation we simulate the construction of a botnet by assuming that the bot code finds and compromises vulnerable computers in the similar way as what a scanning worm does. Fig. 4.2(a) shows the degree distribution for servent bots (client bots always have a degree M, equal to the size of peer list) after the botnet has accumulated 20,000 members. Because the botnet stops growing when it reaches the size of 20,000, the reinfection events rarely happen (only around 600). For this reason, connections to servent bots are extremely unbalanced: more than 80% (4000) of servent bots have degrees less than



(a). Vulnerable population 500, 000 (b). Vulnerable population 20, 000

Figure 4.2: Degree distribution of servent bots (Botnet is constructed via "new infection" and "reinfection" procedure only)

30, while each of the 21 initial servent bots have a degree between 14,000 and 17,500 (the last tiny bar at the bottom right corner of the figure close to X-axis value of 10 represents these 21 servent bots). This is not an ideal botnet. The constructed hybrid P2P botnet is approximately degraded to a centralized botnet where the initial set of servent bots behave as C&C servers.

Vogt *et al.* [121] constructed a super-botnet only with the algorithms that are similar to the "new infection" and "reinfection" procedures presented above. Although authors in [121] showed that their constructed super-botnet is robust, they have an implicit assumption that the super-botnet will have abundant reinfections during its construction period. We believe this assumption is incorrect in a real world scenario—botmasters would want their botnets generating as few as possible reinfections to avoid wasting infection power and being detected by defenders.

To illustrate this argument, we have simulated another botnet scenario where the potential vulnerable population is 20,000 instead of 500,000 used in the previous simulation. The botnet stops propagation after all vulnerable hosts have been infected. Fig. 4.2(b) shows the degree distribution for servent bots in this scenario. When the botnet stops infection

process, overall around 210,000 reinfection events happened. This time, because there are plenty of reinfections, the constructed botnet has a well-balanced connectivity—the degree distribution of all servent bots roughly follows normal distribution, and 80% of servent bots have degrees between 30 and 150.

These simulation experiments show that if a botnet does not have a lot of reinfections, or cannot have reinfection (for example, when the bot program blocks the vulnerable service on an infected host), the aforementioned basic construction procedure is not effective. A botmaster must come up with an additional botnet construction procedure, which is introduced in the following.

4.5.2 Advanced Construction Procedure

One intuitive way to improve the network connectivity would be letting bots keep exchanging and updating their peer lists frequently. However, such a design makes it very easy for defenders to obtain the identities of all servent bots, if one or several bots are captured by defenders.

As introduced in Section 4.4, a botmaster could monitor his botnet easily whenever he wants by issuing a report command. With the detailed botnet information, a botmaster could easily update the peer list in each bot to have a strong and balanced connectivity. The added new procedure is:

- **Peer-list updating:** After a botnet spreads out for a while, a botmaster issues a report command to obtain the information of all currently available servent bots. These servent bots are called *peer-list updating servent bots*. Then, the botmaster issues another command, called *update* command, enabling all bots to obtain an updated peer list from a specified sensor host. Entries in the updated peer list in each bot are randomly chosen from those peer-list updating servent bots.

A botmaster could run this procedure once or a few times during or after botnet propagation stage. After each run of this procedure, all current bots will have uniform and balanced connections to peer-list updating servent bots.

When and how often should this peer-list updating procedure be run? First, this procedure should be executed once shortly after the release of a botnet to prevent defenders from removing all initial servent bots. Second, as a botnet spreads out, each round of this updating procedure makes the constructed botnet have a stronger and more balanced connectivity, but at the same time, it incurs an increasing risk of exposing the botnet to defenders. It is therefore up to a botmaster to strike a comfortable balance. In addition, a botmaster could run this procedure to conveniently reconnect a broken botnet.

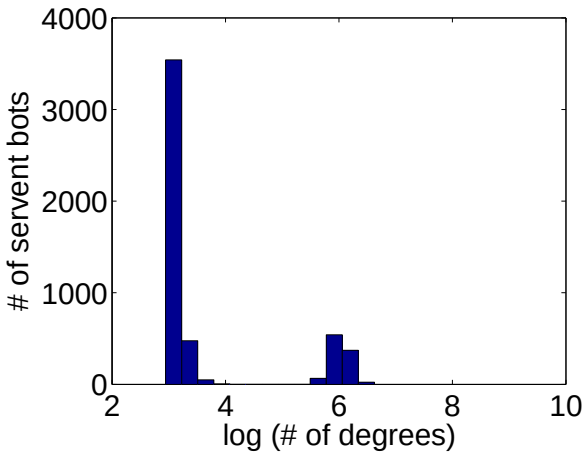


Figure 4.3: Degree distribution of servent bots

Fig. 4.3 shows the degree distribution for servent bots (client bots always have a degree of M) when a botnet uses all three construction procedures. We assume the peer-list updating procedure is executed just once when 1,000 (25% of) servent bots have been infected. This figure shows that although those 4000 servent bots infected after the peer-list updating procedure still have small degrees, the first 1000 servent bots used in peer-list updating have large and balanced connection degrees, ranging from 300 to 500. They form the robust backbone, connecting the hybrid P2P botnet tightly together. On the other hand, the

remaining 4000 servent bots infected after the peer-list updating procedure have connection degrees only around 20 to 30.

4.5.3 Botnet Command Initiation

Comparing Fig. 4.1 with Fig. 1.1(a), we can see that the proposed P2P botnet does not show how its botmaster contacts the botnet to issue commands. Will the flexible service ports on servent bots prevent its botmaster from contacting them? We can answer this question now after introducing the botnet construction procedure.

As used in the simulation experiment, a botnet propagates based on a set of initial servent bots. The botmaster sets their service ports beforehand and thus knows their service ports. After the botnet is released, the botmaster could inject commands through these initial servent bots. After the botmaster issues a report command and gets the first report with the information of service ports of all current servent bots, the botmaster can inject commands through an arbitrarily chosen set of servent bots.

4.6 Botnet Robustness Study

Next, we study the robustness property of a constructed hybrid P2P botnet. Two factors affect the connectivity of a botnet: (1). Some bots are removed by defenders; and (2). Some bots are off-line (for example, due to the diurnal phenomenon [47]). These two factors, even though completely different, have the same impact on botnet connectivity when the botnet is used by its botmaster at a specific time. For this reason, we do not distinguish them in the following study.

4.6.1 Two Robustness Metrics

Since servent bots, especially the servent bots used in peer-list updating procedure, are the backbone connecting a botnet together, we study botnet connectivity when a certain

fraction of peer-list updating servent bots are removed (that is to say, either removed by defenders or off-line).

We present two metric functions to measure robustness. Let $C(p)$ denote the *connected ratio* and $D(p)$ denote the *degree ratio* after removing top p fraction of mostly-connected bots among those peer-list updating servent bots—this is the most efficient and aggressive defense that could be done when defenders have the complete knowledge (topology, bot IP addresses ...) of the botnet. $C(p)$ and $D(p)$ are defined as:

$$C(p) = \frac{\text{\# of bots in the largest connected graph}}{\text{\# of remaining bots}} \qquad \text{(Eq. 4.3)}$$

$$D(p) = \frac{\text{Average degree of the largest connected graph}}{\text{Average degree of the original botnet}} \qquad \text{(Eq. 4.4)}$$

These two metric functions have clear physical meanings. The metric $C(p)$ shows how well a botnet survives a defense action by keeping the remaining members connected together. The metric $D(p)$ shows how densely the remaining botnet is connected together—it exhibits the ability of the remaining botnet to survive a further removal.

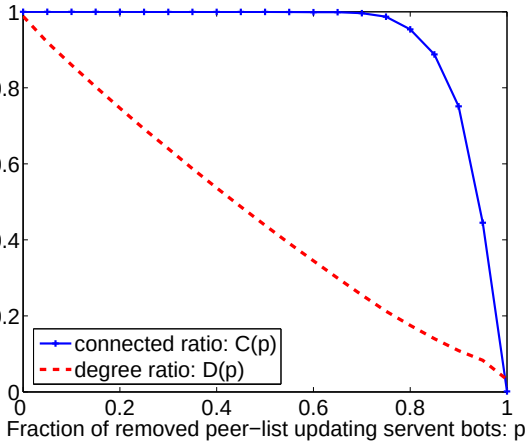


Figure 4.4: Botnet robustness study

Fig. 4.4 shows the robustness of the proposed P2P botnet. The botnet is the one shown in Fig. 4.3 that has a vulnerable population of 500,000 and runs the peer-list updating

procedure only once when 1,000 servent bots are infected. As shown in this figure, if all 1000 peer-list updating servent bots are removed, the botnet will be completely broken. This result shows the importance of the peer-list updating procedure. The botnet will largely stay connected ($C(p) > 95\%$) if less than 700 of those 1000 peer-list updating servent bots are removed, although it has a gradually decreasing connectivity as removal goes on (as exhibited by $D(p)$). This experiment shows the strong resistance of the proposed botnet against defense, even if defenders know the identities of all bots and the complete botnet topology.

4.6.2 Peer-list Updating Procedure

If a botmaster runs the peer-list updating procedure soon after releasing a botnet, he/she will shrink the time window for defenders to shut down the initial servent bots; however, the constructed botnet will rely upon fewer peer-list updating servent bots for its connectivity. Therefore, it is necessary to study how the number of peer-list updating servent bots affects the robustness of a botnet. Fig. 4.5 shows the simulation results in terms of $C(p)$ by varying the number of servent bots used in the peer-list updating procedure from 100 to 2000.

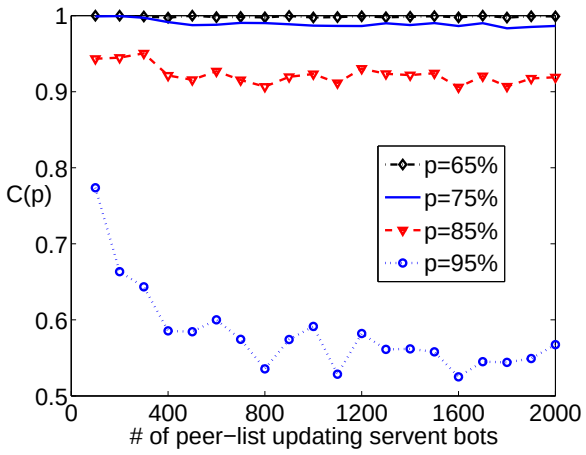


Figure 4.5: Botnet robustness when the peer-list updating procedure runs once with different number of servent bots.

This figure shows that as long as a small fraction of peer-list updating servant bots remain, botnet robustness does not change much when the number of peer-list updating servent bots varies. Of course, if the peer-list updating procedure contains fewer servent bots, it will likewise be easier for defenders to remove most of them, and hence, shut down the botnet.

4.6.3 Mathematical Robustness Analysis

We provide a simple analytical study of the botnet robustness. Assume that each peer list contains M servent bots. It is hard to provide a formula when removing the top p fraction of mostly-connected nodes. However, we could provide the formula of $C(p)$ when *randomly* removing p fraction of peer-list updating servent bots.

As we discussed before, the servent bots not used in peer-list updating procedure have very few extra links besides the M links given by their own peer lists. We simplify the analysis by assuming that each bot in the botnet connects only to peer-list updating servent bots. Then, when we consider removing a fraction of peer-list updating servent bots, more links will be removed compared to the original botnet network. Because of this bias, the analytical formula presented below slightly underestimates $C(p)$ in the case of random removal.

A bot is disconnected from the others when all M servent bots in its peer list have been removed. Because of the random removal, each peer-list updating servent bot has the equal probability p to be removed. Thus, the probability that a bot is disconnected is p^M. Therefore, any remaining bot has the same probability $1 - p^M$ to stay connected, i.e., the mean value of $C(p)$ is (in case of random removal):

$$C(p) = 1 - p^M \qquad \text{(Eq. 4.5)}$$

Fig. 4.6 shows the analytical result from (Eq. 4.5), comparing with the simulation result $C(p)$ of the random removal, and the simulation result $C(p)$ of the removal of top p fraction

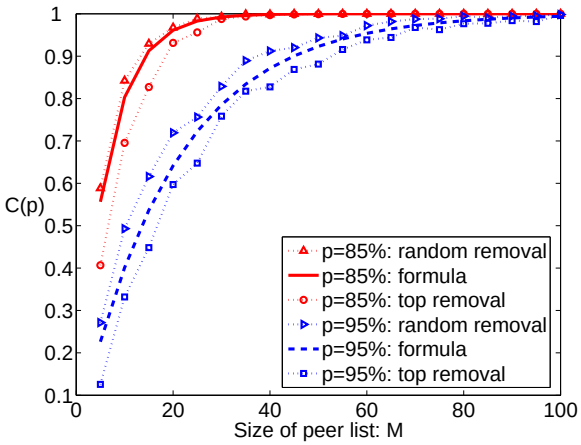


Figure 4.6: Comparison of the analytical formula (Eq. 4.5) and simulation results

of mostly-connected peer-list updating servent bots. The analytical curve lies between those two simulated robustness metrics. It shows that the analytical formula indeed has a small underestimation bias compared with the random removal. Because removing top p fraction will remove more links from the botnet network than a random removal, the simulation results $C(p)$ from the top removal scenario are slightly lower than the derived results from (Eq. 4.5). In summary, this figure shows that, even though the analytical formula (Eq. 4.5) is not very accurate, it provides a good first-hand estimate of the robustness of a botnet.

This figure also shows that the proposed botnet does not need a large peer list to achieve a strong robustness.

For comparison, we can come up with a simple robustness model of today's C&C-based botnets. Suppose a C&C-based botnet has R C&C servers. When defenders have the complete knowledge of such a botnet, they will always remove these R bots first. Thus the botnet robustness metric $C(p)$ is:

$$C(p) = \begin{cases} 1, & < R \text{ bots are removed} \\ 0, & \geq R \text{ bots are removed} \end{cases} \qquad \text{(Eq. 4.6)}$$

the botnet will be shut down if all R C&C server bots are removed, which makes it much less robust than the proposed P2P botnet.

The robustness study presented here is a static study and analysis, considering the robustness of a botnet at any specific moment. In this case, we do not need to consider the botnet infection rate and its spreading speed. If we want to study the dynamics of a botnet when bots are removed gradually, or when bots are removed as the botnet spreads, we will need to consider these two important parameters.

4.7 Defense Against the Proposed Hybrid Peer-to-Peer Botnet

In this section, we discuss how defenders might defend against such an advanced botnet. In addition, we provide simulation studies and mathematical analysis of the performance of botnet monitoring.

4.7.1 Annihilation

First, the proposed hybrid P2P botnet relies on "servent bots" in constructing its communication network. If the botnet is unable to acquire a large number of servent bots, the botnet will be degraded to a traditional centralized botnet (the relationship of these two botnets is discussed in Section 4.2.3), which is much easier to be shut down. For this reason, defenders should focus their defense effort on computers with static global IP addresses, preventing them from being compromised, or removing compromised ones quickly.

Second, as shown in Section 4.5, before a botmaster issues an update command for the first time, a botnet is in its most vulnerable state since it is mainly connected through the small set of initial servent bots. Therefore, defenders should develop quick detection and response systems, enabling them to quickly shut down the initial set of servent bots in a newly created botnet before its botmaster issues the first update command.

The third defense method relies on honeypot techniques. If a botnet cannot detect honeypots, defenders could try to *poison* its communication channel. Defenders let their infected honeypots join the botnet and claim to have static global IP addresses (these honeypots are configured to accept connections from other bots), they will be treated as servent bots. As a result, they will occupy many positions in peer lists of many bots, greatly decreasing the number of valid communication channels in the hybrid P2P botnet. In addition, defenders would know the detailed botnet communication structure and its members through those spying honeypots. With the detailed knowledge of the botnet, defenders could effectively shut it down by cutting off its remaining fragile communication channels.

Another controversial defense approach falls in the category of so-called "good worm" defense [87, 80], or the "cyber-immune system" [103]. Defenders program a "good-purpose" code to exploit the same vulnerability used in a botnet. The code will compromise vulnerable machines in the Internet and patch them. When a machine is already infected by a botnet, the good-purpose code obtains the bot's peer list, cleans the bot code, and then reversely compromises and cleans bots in the peer list. If a cleaned host is contacted by any other bots, the good-purpose code could fire back and clean those bots as well. However, this active defense is in fact another form of Internet attack; it would probably cause more harm than good. Thus it may not be a practical defense in the real world.

As discussed in Section 4.6, the strong robustness of the proposed botnet relies heavily on the peer-list updating procedure. Servent bots used in the peer-list updating procedure form the backbone of the communication network of a botnet. Therefore, the best strategy to disrupt the communication channel of a botnet, if the botnet cannot detect honeypots, is to poison the peer-list updating procedure with the following steps. First, once a honeypot is infected by a bot program, defenders quickly let the bot program infect many other honeypots (for example, by redirecting the bot's outgoing infection traffic to other honeypots). Then, when receiving a report command from the botmaster, all honeypot bots report as servent bots so that they will be used in the peer-list updating procedure. Defenders would

achieve better poisoning defense if they have distributed honeypots and a large number of IP addresses.

When defenders conduct the above poisoning defense, a fraction of servent bots can be treated as being removed from the botnet. The botnet robustness studies presented in Section 4.6 show the effectiveness of such a defense.

4.7.2 Botnet Monitoring Based on Honeypot Techniques

Honeypot is an effective way to trap and spy on malware and malicious activities. Because compromised machines in a botnet need to cooperate and work together, it is particular effective to use honeypot techniques in botnet spying [43, 95], if a botnet cannot detect and get rid off honeypot bots. The third annihilation method introduced above relies on honeypot techniques. In this section, we will introduce botnet monitoring and detection approaches based on honeypot techniques.

4.7.2.1 Botnet Monitoring Based on Spying Honeypots

If a botnet cannot effectively detect honeypots, defenders could let their honeypots join botnets and monitor botnet activities. Based on honeypot bots, defenders may be able to obtain the plain text of commands issued by a botmaster. Once the meaning of the commands is understood, defenders are able to: (1). Quickly find the sensor machines used by a botmaster in report commands. If a sensor machine can be captured by defenders before the collected information on it is erased by its botmaster, they might be able to obtain detailed information of the entire botnet; (2). Know the target in an attack command so that they could implement corresponding countermeasures quickly right before (or as soon as) the actual attack begins.

Another honeypot-based monitoring opportunity happens during peer-list updating procedure. First, defenders could let their honeypot bots claim to be servent bots in peer-list

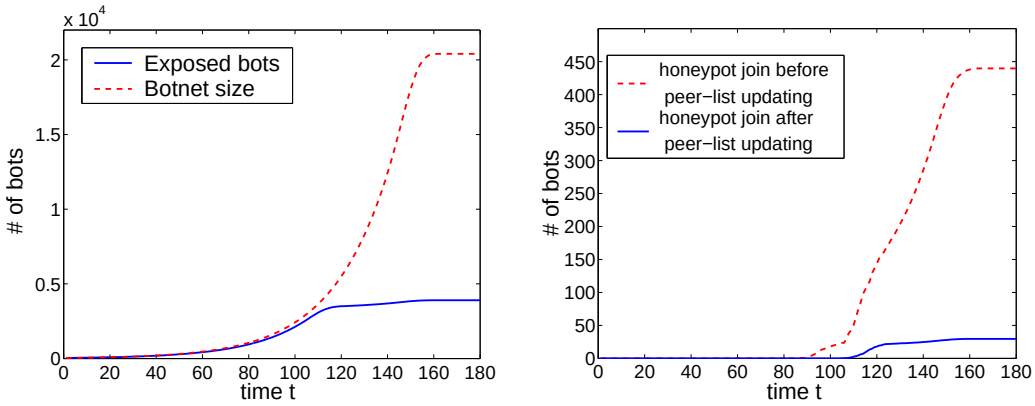
updating. By doing this, these honeypots will be connected by many bots in the botnet; and hence, defenders are able to monitor a large fraction of the botnet. Second, during peer-list updating, each honeypot bot could get a fresh peer list, which means the number of bots revealed to each honeypot could be doubled.

A honeypot could be configured to route all its outgoing traffic to other honeypots; at the same time, the trapped malicious code still believes that it has contacted some real machines. This technique has been used before, such as in [45, 117]. Based on the similar technique, defenders could quickly build up a large number of spying honeypot bots by rerouting the infections sent out from already compromised honeypots to other vulnerable honeypots. In this way, defenders have many spying bots at hand, enabling them to monitor the botnet effectively. Because all of these spying honeypots run the real bot code, a *remote code authentication*[3] cannot enable a botnet or its peer-list updating sensor to detect these spying honeypots.

Upon receiving a botnet report command, a honeypot could have its special program to send back a large amount of identities of fake bots. The information should not be sent from one single IP address. Instead, the honeypot should send out each fake bot's ID and host characteristics with different IP addresses. This approach falls in the category of "Sybil attack" [51]. It is another way to build up spying honeypot bots, to decrease the number of actual core servent bots or the botnet size (if the botmaster stops its botnet growth upon reaching a predefined size). However, this Sybil defense can be defeated if a botnet has a mechanism to conduct remote code authentication, such as using the Oblivious Hashing method [41].

From the above discussion, we can see that by using remote code authentication, a botnet could prevent a defense honeypot from sending information of a large amount of fake bots to its sensor, but it cannot prevent defenders from generating many spying honeypot bots by using real infected honeypots.

[3]Remote code authentication is "the problem of verifying the identity of a remote program." [41]



(a) (b)

Figure 4.7: Botnet monitoring with one honeypot joining as a servent bot. (a) Botnet growth and a honeypot monitoring as one of the initial servent bot; (b) A honeypot monitoring by joining before/after peer-list updating procedure.

For the simulated botnet shown in Fig. 4.3, we conduct another set of simulations where we assume one of its servent bots is a defender's honeypot. We simulate three scenarios: the honeypot joining the botnet as one of initial servent bots; joining the botnet as a servent bot halfway before the peer-list updating procedure (when the botnet accumulates 500 servent bots—the peer-list updating procedure happens when the botnet accumulates 1000 servent bots); and joining the botnet right after the peer-list updating procedure. Fig. 4.7 shows the simulation results averaged over 100 simulation runs. The peer-list updating procedure happens around time t=110 (each bot is assumed to send out 358 scans per unit time to the entire IPv4 space, similar to what Code Red worm did [133]).

If the honeypot is one of the initial servent bots, Fig. 4.7(a) shows that before the peer-list updating procedure, the honeypot could monitor most infected computers since it would appear in most bots' peer lists (as explained in Fig. 4.2). After the peer-list updating procedure, the honeypot could only observe a few more bots because it loses its critical role in the botnet connectivity. Fig. 4.7(b) shows that if the honeypot joins in the botnet halfway before the peer-list updating procedure, it knows on average around 450 bots in the botnet

after the botnet propagation stops; while the honeypot could only know around 30 bots in the botnet if it joins after the peer-list updating procedure.

It could be very hard for a defender's honeypot to be one of initial servent bots, especially botmasters know the risk and select those initial servent bots very carefully. Therefore, we only consider the more realistic monitoring case where honeypots join as servent bots before the botmaster's peer-list updating procedure.

4.7.2.2 Simulation and Analysis of Botnet Monitoring by Multiple Honeypots

It would be interesting to know how many honeypots defenders should set up in order to have an effective monitoring. To study this, we conduct another set of simulations by varying the number of honeypots joining a botnet before the peer-list updating procedure. Fig. 4.8 shows the number of exposed bots after the botnet stops growing as it reaches its desired size of 20,000 (the other simulation settings are the same as the experiment shown in Fig. 4.3). The simulation results are derived by averaging over 100 simulation runs.

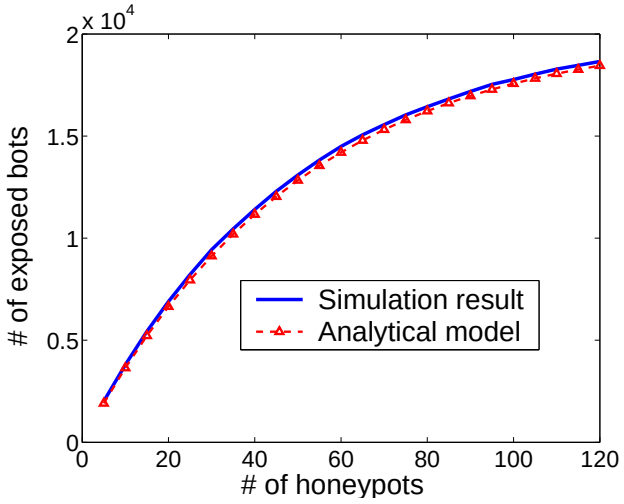


Figure 4.8: Honeypot monitoring by joining as servent bots before the peer-list updating procedure

We can actually derive an analytical model to estimate the mean value of exposed bots, denoted by $E[N_{exposed}]$, when there are n honeypots joining the botnet before the peer-list

updating procedure. Suppose the peer list size is M, the final botnet has I number of bots, and the number of servent bots used in peer-list updating procedure is K. In our simulations, $M = 20, I = 20,000$ and $K = 1,000$.

Before the peer-list updating procedure, because few bots put any of those n honeypots in their peer lists (peer lists are dominated by initial servent bots), we ignore these small number of bots in our analysis, and hence, only consider how many bots have put at least one honeypot in their peer lists after the peer-list updating procedure.

For a specific bot, its peer list contains M servent bots. Since the vast majority of servent bots in peer lists belong to the group of K bots used in peer-list updating procedure, we can assume with little error that all servent bots in any peer list are picked from those K bots.

Thus the probability that the peer list in a specific bot contains none of those n honeypots is:

$$(1 - \frac{n}{K})(1 - \frac{n}{K-1})(1 - \frac{n}{K-2}) \cdots (1 - \frac{n}{K-M}) \qquad \text{(Eq. 4.7)}$$

When $K \gg M$, which is the case in our simulations, (Eq. 4.7) is approximately equal to:

$$(1 - \frac{n}{K})^M \qquad \text{(Eq. 4.8)}$$

which is the probability that this bot will not be exposed to honeypots.

Since the botnet has overall I bots, the average number of exposed bots would be:

$$E[N_{exposed}] = I[1 - (1 - \frac{n}{K})^M] \qquad \text{(Eq. 4.9)}$$

Fig. 4.8 also shows the analytical result derived from the above model (Eq. 4.9), which matches nicely with the simulation results. Because the analytical model ignores the few bots exposed before the peer-list updating procedure, the analytical estimates are slightly smaller than simulation results as exhibited in this figure.

4.7.2.3 Simulation and Analysis of Botnet Monitoring via Darknet Space

The honeypot-based defense methods introduced above assume that compromised honeypots can join in a botnet and continue spying on the botnet activities. Sometimes this requirement may not be satisfied, e.g., when a botnet can quickly detect its honeypot members and remove them from its network [135]. For this reason, we introduce another honeypot-based monitoring technique, which only requires that the botnet could be fooled by a honeypot initially to pass its complete code, including its peer list, to the honeypot.

"Darknet space", or called "black hole", "network telescope", is a chunk of IP space that have no real computers. It is well known that darknet is effective in monitoring Internet malicious traffic [106, 86, 133]. By implementing "honeyd" [21], or an advanced honeypot-based darknet monitor (such as Internet Motion Sensor [17]), defenders may be able to trap a large number of botnet infection attempts. If the bot program cannot detect the darknet monitor and its honeypots initially, and passes its peer list in each infection attempt, defenders could get many copies of peer lists, obtaining the identities and important information (IP addresses, encryption key, service port) of many servent bots in a botnet.

Because the servent bots used in peer-list updating procedure form the backbone of a botnet, we are interested in the fraction of these servent bots that are exposed via a darknet space monitoring. Here we present a simple analytical model for such a monitoring system.

Suppose the darknet begins capturing infection attempts after the peer-list updating procedure. Each captured infection attempt provides one peer list containing M servent bots, and there are K servent bots used in this peer-list updating procedure. For a specific servent bot used in peer-list updating procedure, it has probability p_1 to be exposed by one captured peer list where $p_1 < M/K$. Section 4.6 tells us that most servent bots in the peer list are the ones that have been used in peer list updating procedure. Thus we can estimate p_1 as:

$$p_1 \approx M/K \qquad \text{(Eq. 4.10)}$$

Therefore, if the darknet space captures x infection attempts, the probability that a specific servent bot used in peer-list updating procedure is exposed (through the captured peer lists) is equal to:

$$1 - (1 - \frac{M}{K})^x \qquad \text{(Eq. 4.11)}$$

Based on (Eq. 4.10) and (Eq. 4.11), we can derive the average number of servent bots used in peer-list updating procedure, denoted by $E[N'_{exposed}]$, that will be exposed by x captured infection attempts:

$$E[N'_{exposed}] = K[1 - (1 - \frac{M}{K})^x] \qquad \text{(Eq. 4.12)}$$

Fig. 4.9 shows the simulation results (averaged over 100 simulation runs) and the analytical model results. The analytical results match well with the simulation results but are a bit higher, which is due to the fact that (Eq. 4.10) has slightly overestimated the value of p_1.

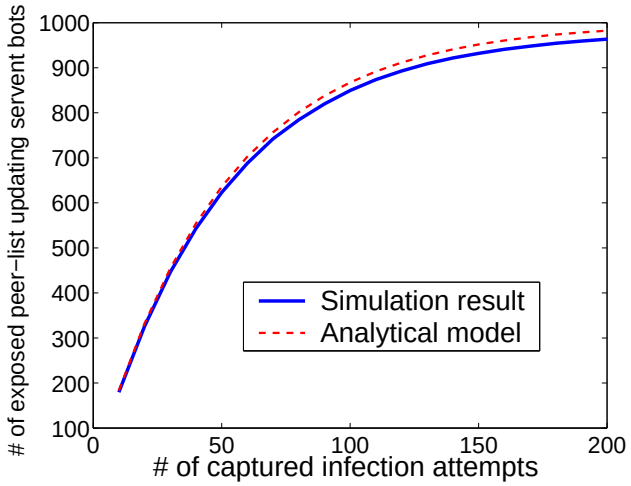


Figure 4.9: Darknet monitoring of servent bots used in peer-list updating procedure

Fig. 4.9 shows that if the darknet can capture 200 copies of peer lists, defenders would be able to know more than 95% of servent bots used in peer-list updating procedure. Thus this darknet based monitoring is an effective way to track the proposed botnet. However, this approach still relies on honeypot techniques. If a bot infection is composed by several

sequential components (which is the case for most current botnets [59]), and a bot passes its peer list to a newly infected host only after the remote host is verified not being a honeypot, the darknet-based monitoring approach would become invalid.

4.7.3 Botnet Detection and Monitoring Without Honeypots

The previous subsection shows that we can propose many effective botnet monitoring approaches based on honeypot techniques. However, as honeypot-based defense systems gradually become popular and widely deployed, botmasters will inevitably develop their botnets to detect honeypots. For this reason, we propose botnet detection and monitoring approaches that do not rely on honeypots.

4.7.3.1 Monitoring Traffic to Botnet Sensor

A possible weakness point of the proposed botnet is its centralized monitoring sensor. If defenders have set up a good traffic logging system, it is possible that they could capture the traffic to a botnet sensor. We call such a monitoring system as a *botnet sensor monitor*. Even though defenders may not be able to capture a botnet sensor before its botmaster destroys the sensor (after completing botmaster's monitoring task), they still could use the captured traffic log to figure out the IP addresses of bots who contacted the sensor in the past. In this way, defenders could get a relatively complete picture of a botnet.

4.7.3.2 Detecting and Monitoring Servent Bots

In the proposed hybrid P2P botnet, servent bots, especially those used in the peer-list updating procedure, are the backbone of a botnet. Fig. 4.3 shows that each servent bot used in the peer-list updating will serve 300 to 500 bots. If a non-server host is infected and serves as one of these servent bots, the host is relatively easy to be spotted by defenders due to the huge increase of traffic in and out of this host. When the number of servent bots

compared to the total botnet population decreases, each of these servent bots must serve a larger number of bots, and hence, is easier to be detected by defenders.

A simple statistical analysis can show this relationship. Denote the number of client bots served by a servent bot as D. For the proposed botnet, D is a random variable. Suppose the peer-list updating procedure is conducted after a botnet finishes its propagation, then all servent bots are used in the updating procedure; and hence, they have evenly distributed connection degrees. Following the same notations as in previous analysis, K is the number of servent bots in a botnet, I is the botnet size, and M is the peer list size. There are $I - K$ client bots in the botnet, each of which connects to M servent bots. Thus there are $(I - K) \cdot M$ connection links initiated from client bots to servent bots. We can derive the mean value of the number of client bots served by each servent bot as:

$$E[D] = \frac{(I - K) \cdot M}{K} = \frac{I \cdot M}{K} - M \qquad \text{(Eq. 4.13)}$$

It is not hard to derive the distribution of D. In the peer-list updating procedure, each client bot is given a randomly chosen peer list by the updating sensor. For any specific servent bot, each client bot has an equal and small probability M/K to connect to the servent bot. Therefore, the random variable D follows "Binormial distribution" with parameters $(I - K)$ and M/K, i.e.,

$$D \sim B(I - K, M/K) \qquad \text{(Eq. 4.14)}$$

4.8 Discussion

From the defense discussion in previous section, we see that honeypot plays a critical role in most defense methods against the proposed hybrid P2P botnet. Botmasters might design countermeasures against honeypot defense systems. Such countermeasures might include detecting honeypots based on software or hardware fingerprinting [102, 44, 21], or

exploiting the legal and ethical constraints held by honeypot owners [135]. Most of current botnets do not attempt to avoid honeypots—perhaps it is simply because attackers have not felt the threat from honeypot defense yet. As honeypot-based defense becomes popular and being widely deployed, we believe botmasters will eventually add honeypot detection mechanisms in their botnets. The war between honeypot-based defense and honeypot-aware botnet attack (Chapter 6) will come soon and intensify in the near future.

For botnet defense, current research shows that it is not very hard to monitor Internet botnets [47, 32, 95]. The hard problem is: how to defend against attacks sent from botnets, since it is normally very hard to shut down a botnet's control? Because of legal and ethical reason, we as security defenders cannot actively attack or compromise a remote bot machine or a botnet C&C server, even if we are sure a remote machine is installed with a bot program. For example, the well-known "good worm" approach is not practical in the real world. The current practice of collaborating with the ISPs containing bot-infected machines is slow and resource-consuming. There are still significant challenges in botnet defense research in this aspect.

From the robustness study in Section 4.6 and the defense study in Section 4.7, we can see that the proposed hybrid P2P botnet makes a future botnet harder to be monitored, but most importantly, makes a botnet MUCH harder to shut down. By replacing a few isolated C&C servers with a significantly larger amount of interleaved servent bots, the proposed botnet greatly increases its survivability.

The proposed hybrid P2P botnet utilizes centralized sensor hosts. This does not make it as weak as a centralized version of botnets. First, sensor hosts are not responsible for botnet command and control communication—their roles are data collection and peer list distribution. If a sensor host is detected and monitored, the botnet could possibly be fully exposed to defenders. However, the botnet will still have its strong survivability as discussed in Section 4.6. In other words, the command and control channel of the proposed botnet is mostly peer-to-peer structured and not affected by sensor hosts. Second, sensor hosts are

disposable. When a botmaster suspects that her senor host is being monitored, he/she can simply discard it and pick another compromised machine as the sensor host.

The proposed hybrid P2P botnet represents only a specific P2P botnet design. In reality, botmasters may come up with some other types of P2P botnet designs. However, we believe this research is still meaningful to security community. The proposed design is practical and can be implemented by botmasters with little engineering complexities. Botmasters will come with a similar design sooner or later, and we must be well prepared for such an attack, or a similar attack, before it happens.

4.9 Summary

In this chapter, we proposed an advanced hybrid P2P botnet. This botnet has many special features which improve its robustness, controllability, and reduce its degree of exposure and detectability. For example, servant bots help efficiently distribute commands; a peer list of fixed number of bots limits the exposure of botnet brought by a compromised bot member; peer list exchange replaces the bootstrap procedure, which is a possible single point of failure during botnet construction; peer list updating procedure balances the connectivity of servant bots; encrypted command and control communication prevents defenders from eavesdropping, and individualized service port prevents defenders from detecting bots based on network flow.

In addition, we introduced two metrics — connected ratio and degree ratio, which are used to evaluate the robustness of botnets under defense. By using simulation-based experiments, the proposed hybrid P2P botnet has shown to be resilient to bot removal defenses.

In the end, we discussed how to fight against proposed botnet from two directions: (1). honeypot-based: our study showed honeypots still play an very important role in botnet mitigation; and (2). non-honeypot-based: in this case, servant bots and sensors set by

botmasters are break points, because they both possess important botnet information and handle huge amount of traffic, which make them relatively easy to be detected.

CHAPTER 5: PEER-TO-PEER BOTNETS COUNTERMEASURES

5.1 Introduction

We believe P2P botnet defense study should be composed with three areas of research: detection, monitoring, and mitigation. Botnet detection has been widely studied by other researchers such as [38,66], and hence, we will not discuss it in this dissertation. Instead, we will exploit and analyze possible solutions for P2P botnet monitoring and mitigation.

In this chapter, we do research on the countermeasures against P2P botnets, especially focus on three defense techniques: index poisoning defense, Sybil defense and passive monitoring. We study these techniques from multiple aspects: presenting the basic ideas, discussing possible attacker's counterattacks, providing mathematical analysis to evaluate their performance, and verifying our analysis using simulation-based experiments, which are done in a P2P botnet simulator developed by us. Besides, upon our study, we are able to obtain one counter-intuitive finding: unlike what people generally think that all P2P botnets are more resilient to defense than centralized botnets, if the index poisoning defense is valid (when the botnet adopts existing P2P protocols and relies on file index to disseminate commands), P2P botnets are equally easy (or hard) to defend compared to traditional centralized botnets.

Actually in research on P2P file-sharing networks, people have long noticed that most P2P protocols are susceptible to index poisoning attack [79] and Sybil attack [51]. Since existing P2P botnets, such as Trojan.Peacomm and Stormnet, directly utilize existing P2P protocols, security defenders could rely on the same principle to conduct index poisoning defense and Sybil defense. Researchers in [57, 64, 49] have pointed out that index poisoning defense and Sybil defense can be used to fight against P2P botnets. However, none of them

Table 5.1: Parameters used in analysis

Symbol		Meaning
Kademlia	m	The number of bits used to represent a node ID or a key.
	k	The maximum number of nodes in a bucket in a routing table.
	Δb	The number of bits improved per step for a lookup.
	c	The number of bits two binary numbers (node ID or key) share in common in their prefixes.
Botnet	N	The number of nodes in a P2P network.
	N_{bot}	The number of bots in a P2P network.
	N_{tz}	The number of bots in the target zone.
	N_{index}	The number of nodes poisoned in the target zone.
	N_{Sybil}	The number of Sybil nodes added to the target zone.
	N_{query}	The number of bots sending out queries for commands.
	l_{tz}	The length of a search path in the target zone.
	$P_{success}$	The probability of a bot getting a real command.

have presented analysis of the performance of these two defense approaches, nor have they discussed how attackers might evade these defenses.

Moreover, for passive monitoring, most of the work are empirical studies. To the best of our knowledge, no mathematical analysis has been provided. But we analyze the effectiveness of passive monitoring, aiming at a specific type of P2P botnet, and are able to give an lower bound for the number of bots that an infiltrated node can monitor.

Before we present our studies of mitigation and monitoring approaches, we will first provide basic background knowledge on the Kademlia P2P protocol, which is the protocol used by the famous Trojan.Peacomm and Stormnet P2P botnets considered in our later discussion.

Notations used in this chapter are summarized and explained in Table 5.1.

5.2 Kademlia Peer-to-Peer Protocol

First, we briefly introduce Kademlia and Kad. For more details about them, please refer to [83, 116, 108].

Kademlia is a distributed hash table (DHT) protocol designed for P2P networks [83], since it is the protocol implemented in Overnet, a P2P network used by Trojan.Peacomm and Stormnet for communication, and such network is our focus in the following sections.

In Kademlia-based networks, each node has a unique node ID, which is represented by an m-bit binary number. Each node has a routing table containing m lists; each list corresponds to each bit of the node ID. Each list is usually referred as a k-bucket, where k is the maximum number of nodes in each list.

Nodes stored in node A's ith k-bucket ($i = 0, 1, ..., m-1$) are the nodes whose node ID must have the first i bits in common with node A's ID, but have a different $(i+1)$-th bit from node A's ID. Fig. 5.1 is an example of a routing table on a node whose ID starts with 1011.

In the distributed hash table preserved in Kademlia-based network, each entry is a <key, value> pair, in which the key is also an m-bit binary number, and the value part stores the corresponding file or node information. Each <key, value> pair is stored on nodes whose node IDs are the closest ones to the key in the network, and the distance is computed as the exclusive or (XOR) of the key and the node ID. The distance between two nodes is computed in the same way.

Kademlia uses iterative routing mechanism. When node A searches for a key, it first finds α nodes which are the closest ones to the key in its routing table, and then initiates lookup queries to these α nodes. Each one of these nodes will send back a response with either the corresponding value part if the <key, value> pair is stored on it, or a certain number of nodes which are the closest ones to the key in its own routing tables if it does not have

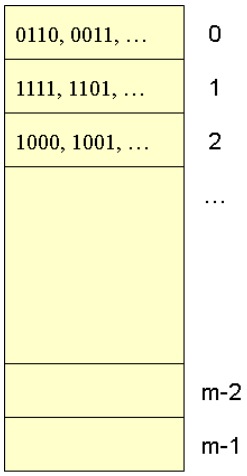


Figure 5.1: Routing table of a node whose ID has m bits and starts with 1011 (For illustration purpose, we only use 4-bit prefix to represent each node). The table contains m lists; each list holds at most k nodes and is called "k-bucket". In the 0th k-bucket, the first bit of each node's ID differs from 1011; in the 1st k-bucket, each node's ID has the same first bit as 1011, but different second bit. In the 2nd k-bucket, nodes share the first two bits with 1011, but have a different third bit. The rest of the k-buckets follow the same manner.

the pair node A is looking for. A lookup query stops when there is a query hit or when it expires.

Besides Kademlia, Kad is another popular DHT protocol for P2P networks. It has been deployed by eMule [5] file-sharing application. However, Kad is based on Kademlia with a slightly different routing table structure and parameter settings, such as the number of bits of a node ID (it is 160 in Kademlia, but 128 in Kad). These differences do not affect our study on Kademlia-based P2P network in the following, so our analysis applies to both Kademlia and Kad networks. In the later discussion, we do not differentiate Kademlia from Kad, unless it is explicitly mentioned otherwise.

5.3 Index Poisoning Defense

5.3.1 Defense Idea

Originally, media companies introduced "index poisoning" defense to prevent illegal distribution of copyrighted content in P2P networks. The main idea is to insert massive number of bogus records into the index. If a peer receives bogus record, it could end up not being able to locate the file (nonexistent location), or downloading the wrong file [79].

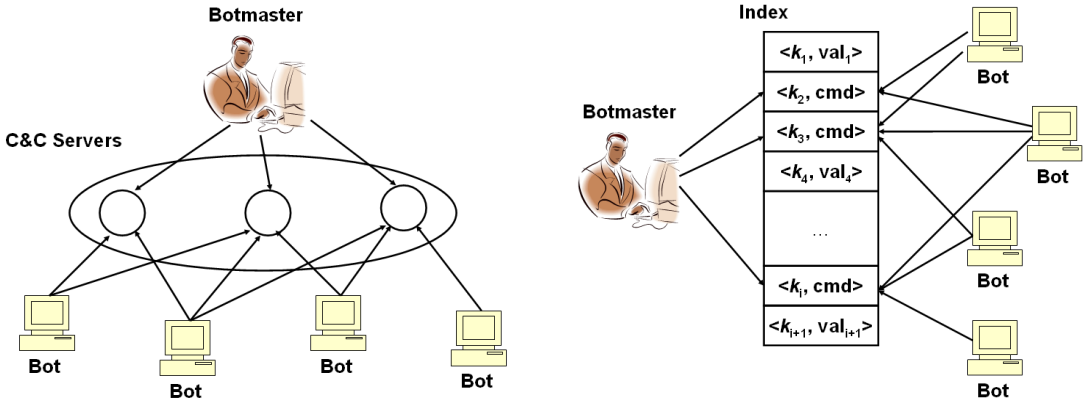
As we discussed in Section 3.4, P2P botnets that implement C&C mechanism of command publishing/subscribing make use of the indices in P2P networks to distribute commands. We refer such botnets as "*index-based*" P2P botnets. Trojan.Peacomm botnet and Stormnet are two typical examples. If defenders are able to figure out the index keys of the botnet command related index records, they can try to "poison" the index by announcing false information under the same keys. If the false information overwhelms the real command content, bots that query and retrieve commands will likely end up obtaining false commands. In this way, the C&C channels of the botnet are disrupted.

We believe there are three reasons that index-based P2P botnets are vulnerable to index poisoning defense.

First, a security defect of P2P protocol itself is the root cause. In most of the P2P networks, there is no central authority to manage the file index, such that any node, no matter benign or malicious, is able to insert records into the index. There is no way to authenticate the identity of the node and content of the records.

Second, with the help of honeypot and reverse engineering techniques, defenders are able to analyze bot behaviors and bot code, and figure out the bot command related index keys.

Third, in some sense, the C&C architecture of this type of P2P botnets is similar to that of the traditional centralized botnets because of the limited number of index keys for command distribution. As shown in Fig. 5.2, in centralized botnets, commands are published at central



(**a**) Centralized Botnet (**b**) Index-based P2P Botnet

Figure 5.2: Similarity of logical C&C structures between traditional centralized botnets and index-based P2P botnets

sites, where bots are going to fetch the commands; on the other hand, in index-based P2P botnets, commands *cmd* are inserted in the index by botmasters under special index keys, such as k_2, k_3 and k_i which are known by bots and queried for retrieving commands.

From the aspect of C&C architecture, index-based P2P botnets logically rely on central points (predefined index keys), while traditional botnets physically rely on central points (predefined C&C servers) for communication. From the aspect of defense, for a traditional C&C botnet, defenders shut down C&C channels by physically removing the C&C servers or blocking access to the servers; while for a P2P botnet, defenders overwhelm real command related records by many bogus records under the same keys (the basic idea of index poisoning technique) to disrupt C&C communication. Therefore, we can draw a conclusion that P2P botnets are not absolutely harder to defend than traditional centralized botnets. If index poisoning defense is valid for a P2P botnet, the P2P botnet is equally easy (or hard) to defend compared with a traditional centralized botnet.

5.3.2 Attackers' Possible Counterattack

Although index poisoning defense is effective, it is still possible for attackers to evade it, if they can eliminate the causes discussed in Section 5.3.1.

Overbot [107], a new botnet protocol designed by Starnberger *et al.*, addressed the second and the third issues. In Overbot, each bot generate its own index key for retrieving command and that key dynamically changes at a certain rate, in addition the communication between bots and sensors (nodes used by botmaster to publish commands) is encrypted. Thus, it is very difficult for defenders to crack or predict the index key, even though defenders are able to do it for one single bot, it is still not helpful, because each bot has different index key. However, for the same reason, sensors have to publish a <key, command> pair for each bot they know periodically, which increases the sensors' workloads and makes them more suspicious. In other words, the advantages of Overbot come with the lack of scalability and increase of detectability.

Now we present a novel and realistic method that attackers might use to deal with index poisoning defense—an authentication enforcement for command generation and index manipulation. It addresses the first cause of index-based P2P botnets being vulnerable to index poisoning (Section 5.3.1). In this approach, only botmasters can insert records to the command index preserved on bots. Bots can only query to fetch commands.

To realize the authentication, a botmaster generates a pair of public/private keys $< K^+, K^- >$, and hard-codes the public key K^+ into the bot code. Later, when the botmaster wants to issue a command m under key k, he/she can insert a record $< k, m, K^-(H(m)) >$ instead of $< k, m >$ into the index on a bot, saying bot A, where $H(m)$ is the hash value of m (i.e., the command is signed by the botmaster). Bot A can decide if the record is created by its botmaster or not by using the public key K^+ to verify the signature. If the signature is authentic, bot A stores this record in the index and waits for others to query, otherwise it discards the fake one. In this way, the index on a bot will not be polluted.

In addition, this authentication mechanism can prevent the spread of false commands. Even if defenders manage to store entries with forged commands in the index on controlled peers (e.g., honeypots infected by a captured bot binary), bots can verify the authenticity of received commands using the public key and disregard the false ones.

Most of existing P2P protocols have not implemented this kind of authentication mechanism. Thus in order to deploy it, attackers need to modify the existing P2P protocols, which implies that this counter index poisoning technique can only be applied to bot-only P2P botnets because the botnets cannot join existing P2P file-sharing networks anymore.

5.3.3 Analytical Study

In this section, we give an analytical study on performance of index poisoning defense against index-based P2P botnets. Our target is a P2P botnet like Trajon.Peacomm botnet and Stormnet that has implemented Kademlia-based distributed hash table (DHT) protocol for C&C communication. Similar study can be conducted on P2P botnets utilizing other protocols.

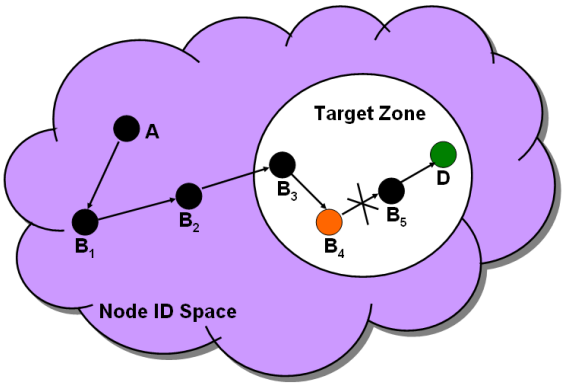


Figure 5.3: A search path for a key, where node A is the initiator and node D is the destination. On this path node B_4 could be a node targeted by defenders to interrupt the search.

As introduced in Section 5.2, in a Kademlia-based DHT, each entry is a <key, value> pair, and each pair is stored on at least one node whose node ID is closest to the key in the network. If defenders want to pollute a P2P botnet's index records under key K, they need to contact nodes (poisoned nodes) whose IDs are close to K, and store pairs like $<K$, false value> on them. In this way, when a bot queries for key K to retrieve commands, those poisoned nodes will have a good chance to appear on the search path and return false query

value, and hence, prevent bots from reaching nodes who possess the real commands. As illustrated in Fig. 5.3, a bot node A initiates a lookup for index key K, the search path is supposed to go through node B_1, B_2, B_3, B_4 and B_5, until it reaches node D who has the pair <K, command>. If a pair <K, false command> has been added in the index on node B_4, when the lookup message reaches node B_4, the node would return the false command and terminates the search.

We assume that node IDs are uniformly distributed over the entire Kademlia ID space, which is supported by the study in [108]. Suppose defenders choose N_{index} nodes whose IDs agree at least the first c bits in common with K. We call the zone around K the "*target zone*". Only when a lookup path enters the target zone, is it possible that a poisoned node will be chosen, return a search result and terminate the search. Let x be the probability of choosing a poisoned node in one lookup step, then $1 - x$ is the probability of not choosing one. Therefore the probability of a bot obtaining the real command is

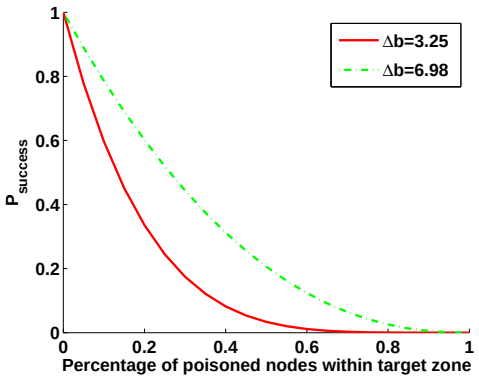
$$P_{success} = (1 - x)^{l_{tz}} \qquad \text{(Eq. 5.1)}$$

where l_{tz} is the length of a search path within the target zone.

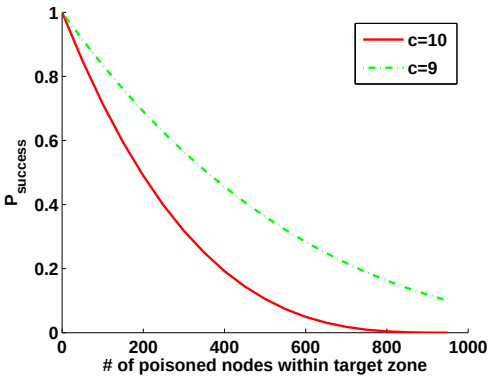
When a peer initiates a lookup for a key, in general, the expected number of steps required to perform a lookup is given as follows [116]:

$$l = \frac{\log_2 N}{\Delta b} \qquad \text{(Eq. 5.2)}$$

where N is the size of the network. We assume all the nodes are bots, so $N_{bot} = N$ in this case. And Δb is the number of bits improved per step, which depends on the structure of the routing table. Thus within the target zone, $l_{tz} = log_2 N_{tz}/\Delta b$. Since node IDs are uniformly distributed, the number of nodes in the target zone is $N_{tz} = N/2^c$, and $x = N_{index}/N_{tz}$. The



(**a**) Index poisoning with different Δb
$N = N_{bot} = 980000$, $c = 4$



(**b**) Index poisoning with different c
$N = N_{bot} = 980000$, $\Delta b = 3.25$

Figure 5.4: Performance of index poisoning defense technique illustrated by numerical results

complete formula to calculate $P_{success}$ is

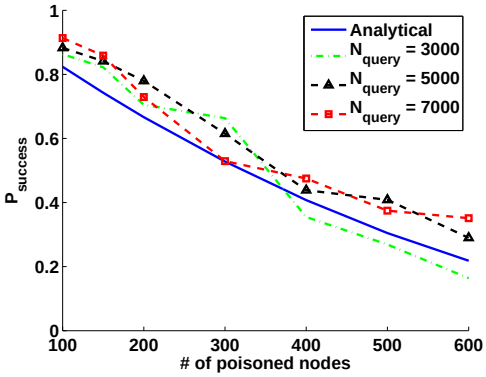
$$P_{success} = (1 - \frac{2^c \times N_{index}}{N})^{(\log_2 N - c)/\Delta b} \qquad \text{(Eq. 5.3)}$$

According to Equation (Eq. 5.3), the performance of index poisoning technique depends on four parameters. We have provided numerical results in Fig. 5.4 to show their impacts on $P_{success}$ by changing the parameters.

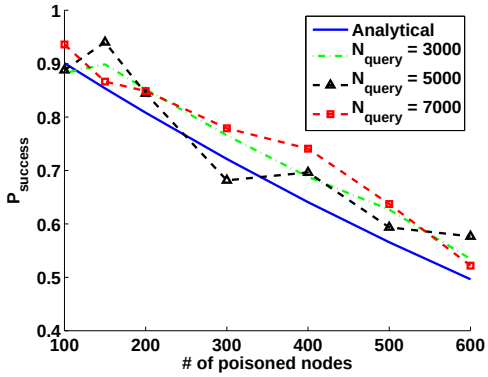
Fig. 5.4(a) illustrates that a botnet would be more robust to index poisoning defense, if for each lookup more bits can be improved, i.e., the average length of search path is short. When search path is short, poisoned nodes have less chance to be chosen along the path[1].

It is shown in Fig. 5.4(b) that in order to achieve better performance, defender could choose a larger c, i.e. choosing nodes that are closer to the command related key to poison. However it is not always a good idea to choose a large c, because we want to have at least one step along the lookup path falls in the target zone, otherwise bots can directly get commands without going through a poisoned node. In other words, $l_{tz} \geq 1$, i.e., $c \leq \log_2 N - \Delta b$. In our case, $N = 980000$, $\Delta b = 3.25$, so $c \leq 16.7$, and for the setup $c = 9$ and $c = 10$ used in Fig. 5.4(b), l_{tz} is 3.35 and 3.05 respectively.

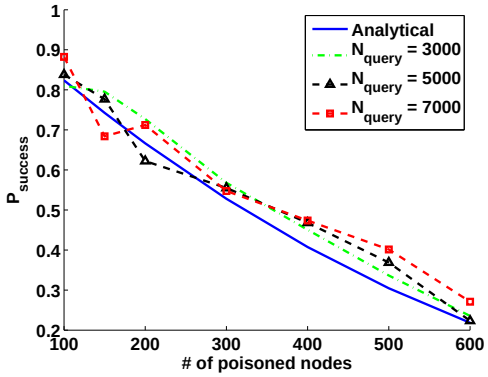
[1]The value of Δb was estimated in [116]. 3.25 is the worst case, while 6.98 is the best case.



(**a**) N_{bot}=N=10000, m=16



(**b**) N_{bot}=N=20000, m=16



(**c**) N_{bot}=N=10000, m=160

Figure 5.5: Comparison between analytical and simulation results of $P_{success}$ for index poisoning defense. The simulated P2P botnet is Kademlia-based with a $D(1, 1, 8)$ routing table structure, and $c = 3$.

The size of the network would also affect the performance of index poisoning defense. However, it does not matter that much, since given a fixed percentage of poisoned nodes in the target zone, it can barely change l_{tz} due to the log_2 operator ($log_2 980000 = 19.90$ and $log_2(12000 \times 2^8) = 22.29$)[2].

5.3.4 Simulation Evaluation

To evaluate the accuracy of our analysis, we develop a P2P botnet simulator based on OverSim [33], an open source P2P simulator. The P2P botnet we simulate employs Kademlia protocol for the C&C communication.

In [116], Stutzbach *et al.* defined a system $D(b, r, k)$, which uses b-bit symbols with r-bit resolution and k-buckets, to represent the routing table structure of a Kademlia-based DHT protocol. According to this definition, the routing table structure implemented in our simulator can be denoted as $D(1, 1, 8)$ (i.e., $b = 1$, $r = 1$, $k = 8$), which is consistent with the basic Kademlia design. Correspondingly, the average bits improved per lookup step in our simulated system is $\Delta b = 4.41$[3] To reduce the computation time, we set m to be 16 instead of 160 which is the default setting in Kademlia protocol. Experiments on comparing the performance of index poisoning defense with different values of m (Fig. 5.5(a)(c)) show that its value does not matter.

We consider two different sizes of such botnets with N=10000, and N=20000, respectively. The parameters we change are N_{query}, the number of bots who queries for commands and N_{index}, the number of bots whose indices has been poisoned. The node IDs of these N_{index} poisoned nodes share at least the first $c = 3$ bits with a given command related key. In each simulation run, every bot in the set of N_{query} query bots looks for the command once; and we calculate $P_{success}$ based on how many of them actually obtain the real command. To derive the average value of $P_{success}$, we conducted at least 20 simulation runs for each botnet configuration.

Fig. 5.5 shows the experiment results comparing to the analytical result obtained from our analysis. According to Equation (Eq. 5.3), $P_{success}$ does not depend on N_{query}. Therefore, only one curve is plotted as the analytical result (the solid blue line in the figure). As we

[2]An estimate was given in [116] that the Kad network has around 980,000 concurrent peers. Authors of [108] claimed that the population of peers in Kad network is between $12{,}000\times2^8$ and $20{,}000\times2^8$.

[3]Please refer to the paper [116] for the detailed formulas to compute Δb given the routing table structure $D(b, r, k)$.

can see, the analytical result matches with simulation results of $P_{success}$ with around 10% of errors. Fig. 5.5(c) plots the results from another simulation with the same settings as Fig. 5.5(a), except that $m = 160$. According to our analysis (Eq. 5.3), $P_{success}$ does not depend on the value of m. Fig. 5.5(c) confirms this conclusion.

5.4 Sybil Defense

5.4.1 Defense Idea

In a normal P2P file-sharing network, "Sybil attack" is referred as the forging of multiple identities by attackers to subvert the reputation system [51]. The reason of P2P networks being vulnerable to Sybil attack is that peers can join the network without authentication or validation of their identities. It is an inherent vulnerability for most P2P networks and protocols [109, 123].

For the same reason, an index-based P2P botnet that implements a traditional P2P protocol will also be susceptible to Sybil-based defense as well. With the knowledge of index keys used for command distribution, defenders can add Sybil nodes (such as honeypots) into the botnet to re-route or monitor the command related traffic. How to set up Sybil nodes depends on the P2P system implementation. In an unstructured P2P network, in order to capture more botnet traffic, defenders will set up Sybils to be peers with more important roles, e.g. setting up Sybil nodes as ultrapeers in Gnutella because only ultrapeers are allowed to forward messages. In a structured P2P network, such as Kad, the node IDs of Sybil nodes should not be chosen randomly, but be close to a known command related index key, as discussed in [64, 49]. In this way, command query traffic for the key will go through the Sybil nodes with a high probability according to the Kad's routing algorithm. We call such defense "*targeted*" Sybil defense.

For defenders, the cost for Sybil defense is usually higher than index poisoning defense. This is because either a physical or a virtual machine is needed to set up a Sybil node; in

other words, more Sybil nodes require more computer resources, while publishing different records to poison index can be done by a single node.

5.4.2 Attackers' Possible Counterattack

Similarly, approaches used for protecting today's P2P networks from Sybil attack may also work for botmasters to prevent defenders from infiltrating their P2P botnets using Sybil nodes. Here we briefly introduce possible counterattack methods.

In Kademlia-based P2P networks, a node ID can be constructed by hashing the node's IP address as what Chord does [112], rather than being randomly generated by a joining node itself like what Kad does [108]. If the network uses a node's IP address to generate the node ID, Sybil nodes will not be able to choose any IDs they want. When a botmaster applies this scheme in his/her P2P botnet, defenders cannot target a specific key to set up their Sybil nodes. In this case, Sybil nodes are just randomly added into the botnet, which is referred as "*random*" Sybil attack. This kind of Sybil defense is much less effective than targeted Sybil defense as shown in the next section.

Furthermore, caching technique [83], which was meant to solve "hot spots" problem, can also be utilized by a P2P botnet to reduce the effectiveness of Sybil defense. Because the command related index records will be stored not only on bots that were chosen at the beginning by their botmaster (e.g., unstructured network) or according to the protocol (e.g., structured network), but also on bots that may not be easily identified. Thus even targeted Sybil defense cannot cover all the bots that possess the command information.

5.4.3 Analytical Study

Now we analyze Sybil defense on the same type of P2P botnets as in Section 5.3.3, Kademlia-based P2P botnets. The notations have the same meaning unless explicitly mentioned.

If node IDs can be chosen randomly, defenders can create special N_{Sybil} Sybil nodes, whose node IDs share at least the first c bits with an index key K, and add them into the botnet. Once a Sybil is chosen on the path of a command lookup, it can re-route the message or return a false command and terminate the search, and hence, prevent the query bot from obtaining the real command.

As we can see, Sybil defense shares the same defense principle with index poisoning defense. They both try to manipulate the command lookup path, as shown in Fig. 5.3. Sybil defense achieves this manipulation by adding new special nodes (controlled by defenders) to the network, i.e., node B_4 in Fig. 5.3 is a Sybil node added by defenders, while index poisoning defense achieves this by poisoning the nodes (bots probably) already in the network.

Following the same analysis procedure as what we did in Section 5.3.3, the probability of a bot successfully getting the real command $P_{success}$ can be calculated using Equation (Eq. 5.1), except that x becomes the probability of choosing a Sybil at each step along the search path within the target zone, which is $N_{Sybil}/(N_{Sybil} + N_{tz})$. So

$$P_{success} = (1 - \frac{N_{Sybil}}{N_{Sybil} + N_{tz}})^{l_{tz}} \qquad \text{(Eq. 5.4)}$$

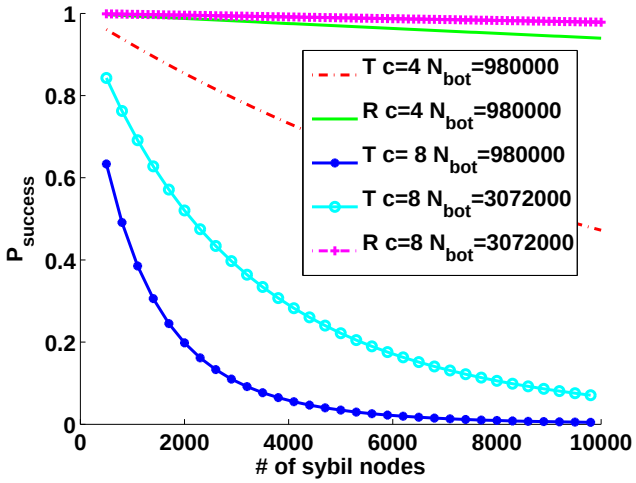
where $l_{tz} = log_2(N_{Sybil} + N_{tz})/\Delta b$, and $N_{tz} = N_{bot}/2^c$.

Differing from what used in the index poisoning defense analysis, the size of the network used in Sybil defense analysis is not the number of the bots, but the total number of bots and Sybil nodes, i.e., $N = N_{bot} + N_{sybil}$, since Sybil nodes added by defenders are not real bots.

When a verification mechanism for node ID is applied in the botnet (Section 5.4.2) such that defenders can only conduct random Sybil defense, the whole network becomes the target zone, i.e., $N_{tz} = N_{bot}$. Simply substituting N_{tz} in Equation (Eq. 5.4) by N_{bot}, we can get the following formula to compute $P_{success}$ in this "random" Sybil defense.

$$P_{success} = (1 - \frac{N_{Sybil}}{N_{Sybil} + N_{bot}})^{log_2(N_{Sybil}+N_{bot})/\Delta b} \quad \text{(Eq. 5.5)}$$

It is shown in Fig. 5.6 that under the same circumstance targeted Sybil defense outperforms the random one. This is because in the former case, Sybil nodes with specially chosen IDs have more chances to appear along a search path than those in the latter case. With limited resources that defenders may use to launch Sybil defense, if the Sybil nodes are closer to the key K (i.e., larger c), the defense performance would be better. Furthermore, Sybil defense is more effective if the network is smaller.

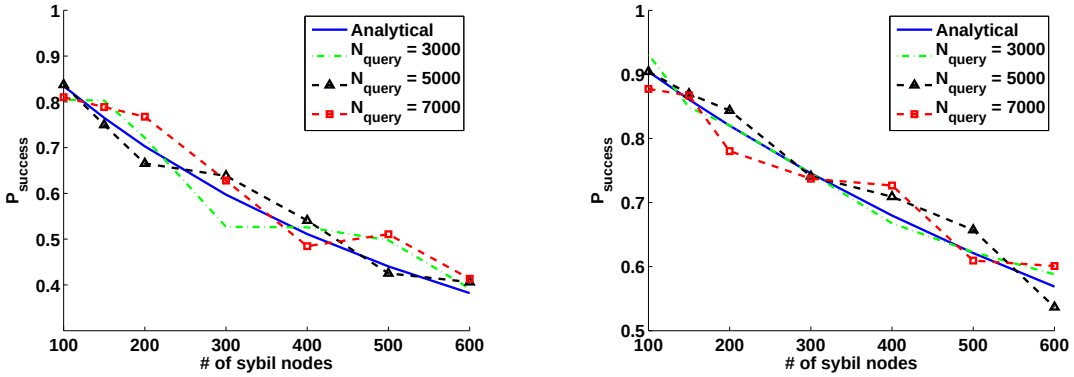


T — targeted Sybil defense, R — random Sybil defense, $c = 4$, $\Delta b = 3.25$

Figure 5.6: Performance of Sybil defense technique illustrated by numerical results

5.4.4 Simulation Evaluation

We use simulation experiments as well to verify our analysis. The network settings and parameter configurations are the same as those used in Section 5.3.4. The botnet is a Kademlia-base P2P network, with a $D(1, 1, 8)$ routing table structure and node ID or key of 16 bits long (m=16).



(**a**) N_{bot}=10000 (**b**) N_{bot}=20000

Figure 5.7: Comparison between analytical and simulation results of $P_{success}$ for Sybil defense. The simulated P2P botnet is Kademlia-based with a $D(1, 1, 8)$ routing table structure, and $c = 3$.

We assume N_{Sybil} Sybil nodes, whose node IDs share at least the first 3 bits (c=3) with a given key, are added by defenders during the construction of the botnet. When the botnet is built up, the whole network has $N = N_{bot} + N_{Sybil}$ nodes, and N_{qeury} bots will start querying for commands. Again, we use 20 simulation runs to obtain the average simulation results. The simulation results along with the numerical results are plotted in Fig. 5.7, which show that our analysis is consistent with the simulations.

5.5 Passive Monitoring of Peer-to-Peer Botnet

Botnet monitoring is an important component in the overall botnet defense. A good monitoring could collect valuable information about the botnet under observation, such as the size of the botnet, the unique features of the botnet network traffic, and the identities of bots, etc.

Monitoring systems can be classified as either active or passive. Active monitoring usually starts with one or a couple of known bots within the network. By actively contacting these bots, defenders could get to know the identities and information of more bots, and make contacts with those newly discovered bots in the next round. The monitoring is done actively

and iteratively until no more unknown bots can be discovered. Passive monitoring is carried out by Sybil nodes put by defenders in the botnet. Unlike active monitoring, these nodes do not actively contact other nodes in the network; they only perform the routine tasks like other normal nodes, such as forwarding traffic and responding to queries, like normal nodes. Nodes that have contacted the monitoring nodes are recorded for further analysis. Passive monitoring has the advantage of being stealthy, and hence, harder for botmasters to detect and remove those monitoring nodes from their botnets.

In this section, we provide mathematical analysis of the effectiveness of passive monitoring. In other words, we would like to figure out how many bots in a P2P botnet that a passive monitoring node can observe after a certain time period. In this section we address this problem in a Kademlia-based P2P botnet as well.

5.5.1 Analytical Study

Suppose defenders have set up one passive monitoring node in a P2P botnet. We want to estimate the number of bots that have a monitoring node in their routing tables, denoted as $N_{routing}$. According to Kademlia protocol, a node would contact nodes in its routing table from time to time because of query or routing table refresh activities. Therefore, $N_{routing}$ is the lower bound for the number of bots that can be observed by a monitoring node.

In a Kademlia-based P2P botnet with N nodes, we denote each node as B_i, where $i = 1, 2, ..., N$, and the time of node B_i joining the network is denoted as t_i, where we assume $t_i < t_j$, if $i < j$, and $t_i \neq t_j$, if $i \neq j$, i.e, no two nodes join the network at the same time. Moreover, when node B_i joins the network, the current size of the network is denoted as N_i. Because of the way we index the nodes, we can easily know that $N_i = i$.

To compute the number of nodes who have a specific node B_i in their routing tables, we need to consider two types of nodes: the nodes joining the network before B_i, referred as $Nodes_{before}$, and the nodes joining the network after B_i, referred as $Nodes_{after}$.

When node B_i joins the network, there are already $N_{i-1} = i-1$ nodes in the network. We can classify these N_{i-1} nodes into m groups. The c-th group ($c = 0, 1, 2, \cdots, m-1$) contains the nodes whose IDs share the first c bits with node B_i's ID but differ at the $(c+1)$-th bit. Because node IDs are uniformly distributed, the number of nodes in the c-th group is $N_{share}(c) = N_{i-1}/2^{c+1}$. If a node in the c-th group whose c-th bucket is not full (i.e., $N_{share}(c) \leq k$), it will add node B_i in this bucket, otherwise it will not contain node B_i in its routing table. As c increases, the size of c-th group is monotonously decreasing. When $0 \leq c < c_0$ where $c_0 = \lceil \log^{N_{i-1}/k} - 1 \rceil$ (c_0 is obtained by letting $N_{share}(c) = k$), $N_{share}(c) > k$, and hence, we do not need to consider nodes in these groups. Therefore the number of $Nodes_{before}$ who would add node B_i into their routing tables can be calculated as follows:

$$N_{before}(i) = \sum_{c=c_0}^{m-1} \frac{N_{i-1}}{2^{c+1}}, \qquad \text{(Eq. 5.6)}$$

where $c_0 = \lceil \log^{N_{i-1}/k} - 1 \rceil$, which is obtained by letting $N_{share} = k$.

After node B_i has joined the network, for the nodes joining in later on, they may add node B_i into their routing tables as well. Let's consider a node $B_j, i < j$, the probability of these two nodes' IDs sharing the first c bits but differing at the $(c+1)$-th bit is

$$P_{share}(c) = \frac{2^{m-(c+1)} - \frac{N_j}{2^{c+1}}}{2^m - N_j} = \frac{1}{2^{c+1}}, c = 0, 1, ..., m-1. \qquad \text{(Eq. 5.7)}$$

Suppose node B_i and node B_j share the first c bits but differ at the $(c+1)$-th bit in their IDs, there are $N_{share}(c) = N_{j-1}/2^{c+1}$ candidates for node B_j to pick and add to its c-th k-bucket, and node B_i is in this candidate set. We can consider two possible scenarios. When $N_{share}(c) > k$, node B_j randomly picks k nodes from the candidate set to put in its routing table; when $N_{share}(c) \leq k$, all the nodes in the candidate set will be chosen. Therefore, the probability of node B_j adding node B_i into its routing table is

$$P_{add}(c) = \begin{cases} \frac{k}{\frac{N_{j-1}}{2^{c+1}}}, & \frac{N_{j-1}}{2^{c+1}} > k \\ \\ 1, & \frac{N_{j-1}}{2^{c+1}} \leq k \end{cases} \quad \text{(Eq. 5.8)}$$

Let $c_1 = \lceil \log^{N_{j-1}/k} - 1 \rceil$, i.e., $N_{j-1}/2^{c_1+1} = k$, we can rewrite Equation (Eq. 5.8) and get Equation (Eq. 5.9).

$$P_{add}(c) = \begin{cases} \frac{k}{\frac{N_{j-1}}{2^{c+1}}}, & c < c_1 \\ \\ 1, & c \geq c_1 \end{cases} \quad \text{(Eq. 5.9)}$$

Therefore, the number of $Nodes_{after}$ that would have node B_i in its routing table can be calculated as follows:

$$N_{after}(i) = \sum_{c=0}^{m-1} P_{share}(c) \times P_{add}(c) \quad \text{(Eq. 5.10)}$$

For a specific node B_i, the total number of nodes having it in their routing tables is

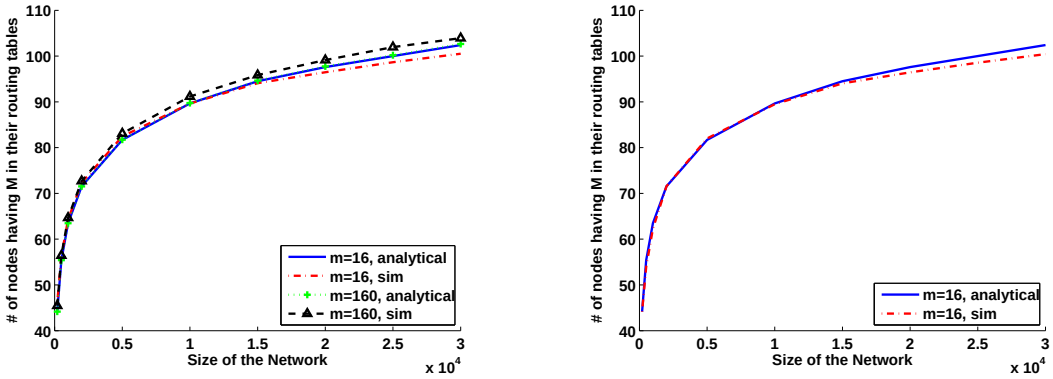
$$N_{routing}(i) = N_{before}(i) + N_{after}(i) \quad \text{(Eq. 5.11)}$$

and the average number of nodes that have a monitoring node in their routing tables is

$$\overline{N}_{routing} = \frac{1}{N} \sum_{i=1}^{N} N_{routing}(i) \quad \text{(Eq. 5.12)}$$

5.5.2 Simulation Evaluation

Still we simulate the same Kademlia-based P2P botnet as the one in Section 5.3.4 and Section 5.4.4. We carry out two types of experiments: network without churn and with



(**a**) Without churn (**b**) With churn

Figure 5.8: Comparison between analytical and simulation results of $\overline{N}_{routing}$. The simulated P2P botnet is Kademlia-based with a $D(1, 1, 8)$ routing table structure.

churn, where churn refers to the network dynamics caused by nodes' joining and leaving activities.

In the network without churn, we consider once the botnet is constructed, the botnet is stable, i.e., no nodes will leave the network and no more nodes will join the network as well. $\overline{N}_{routing}$, the average number of nodes which have a given monitoring node in their routing tables for different scales of network is shown in Fig. 5.8(a). We can see that our analysis precisely estimates $\overline{N}_{routing}$.

However, in the real world, the churn does exist in P2P botnets. To make our experiment more realistic, we introduce the churn events in our simulations. In our simulated P2P network, node joining and node leaving events will happen, and we assume the time interval between two churn events t_{churn} follows a truncated normal distribution (i.e., $t_{churn} \sim N(\mu, \sigma^2)$ and $t_{churn} > 0$). In order not to favor any one of the node joining and node's leaving, we set, when a churn event happens, the probability of it being a node's joining P_{in} and of it being a node's leaving P_{out} to be the same, i.e., they are both 50%.

Fig. 5.8(b) shows our simulation results when considering churn. In our experiments, all simulations run for the same amount of time (30,000 unit time) and t_{churn} follows the same distribution ($\mu = 15$ and $\sigma = \mu/3$). As a result, in each simulation, there are around

2000 node joining/leaving events. If the size of the network is small, only a small fraction of original bot nodes (e.g., the first N nodes) still exist in the network when the simulation ends. But in a relatively large network, a large fraction of original nodes still exist in the network. For example, when N=200, there are around 4%-5% of the first 200 nodes still in the network at the end of the simulation; while when N=30,000, 96% of the original 30,000 nodes remain in the network. From another perspective, we can view this phenomenon as the illustration of monitoring performance under different churn intensities. Since in our experiments, we cover the sizes of network from 200, 500, ... 30,000, we have considered the monitoring performance under different churn intensities. As what is shown in Fig. 5.8(b), our analysis can still well evaluate $\overline{N}_{routing}$ even with churn.

5.6 Others Countermeasures

In the following, we present several general ideas to counter P2P botnets.

5.6.1 Detection

Being able to detect bot infection can stop a new-born botnet in its infant stage. Signature-based malware detection is effective and still widely used. But anti-signature techniques, such as polymorphic technique [72], make it possible for malware to evade such detection systems. Therefore, instead of doing static analysis, defenders start considering dynamic information for detection. The system proposed by Gu *et al.* [59] is one based on dynamic pattern matching.

Anomaly detection is another direction, since bots usually exhibit different behaviors from legitimate P2P users, such as sending queries periodically, always querying for the same content, or repeatedly querying but never downloading.

Besides, distributed detection is another approach, such as a self-defense infrastructure presented in [130], and two approaches against ultra-fast topological worms in [128].

5.6.2 Monitoring

Monitoring botnets help people better understand their motivations, working patterns, evolution of designs, etc. There are two effective ways to conduct P2P botnet monitoring.

For parasite and leeching P2P botnets, we can choose legitimate nodes in the host P2P networks as sensors for botnet monitoring. Usually sensors are peers that play important roles in the network communication, such as ultrapeers in Gnutella networks, thus more information can be collected. In DHT-based P2P networks, the search path of a specific key is relatively fixed, even if the search starts at different nodes. So the sensor selection depends on the monitoring targets and routing algorithm implemented in the system.

Honeypot techniques [105] are widely used for botnet monitoring. The way to set up honeypots in a P2P botnet is similar to choosing sensors. The difference is that honeypots are hosts added to the network on purpose by defenders, while sensors are chosen from the nodes who are already in the network.

5.6.3 Shutdown

The ultimate purpose of studying botnets is to shut them down. We can either 1) remove discovered bots, or 2) shut down C&C channels of botnets.

Botnet construction relying on bootstrapping is vulnerable during its early stage. Isolating or shutting down bootstrap servers or the bots in the initial list that are hard-coded in bot code can effectively prevent a new-born botnet from growing into a real threat.

P2P botnets can also be shut down or partially disabled by removing bot members. There are two modes of bot removal: random and targeted. The former means disinfecting the host whenever it is identified as a bot. The latter means removing critical bots, such as the ones that are important for C&C communication, when we have the knowledge of the topology or C&C architecture of a P2P botnet. We have studied these two types of removal on the proposed hybrid P2P botnet in Chapter 4.

Shutting down detected bots is slow in disabling a botnet and sometimes impossible to do (e.g., you have no control of infected machines abroad). So a more effective and feasible way is to interrupt botnet C&C communication such that bots cannot receive orders from their botmaster. This approach has been carried out well for centralized botnets through shutting down the central C&C sites, but is believed to be more difficult to do for P2P botnets.

However, we find that this general understanding of "*P2P botnets are much more robust against defense*" is misleading. In fact, index-based P2P botnets are as vulnerable as centralized botnets, if the counter defense methods we presented in Section 5.3.2 and 5.4.2 are not implemented. Index poisoning defense (Section 5.3) and Sybil defense (Section 5.4) can be quite effective to fight against such botnets.

5.7 Discussion

It is worthy to point out that the search process we discussed in Section 5.3.3 and Section 5.4.3 can be performed in two different manners – iterative and recursive. Let us use the scenario presented in Fig. 5.3 to explain the difference between these two search modes: the iterative search route would be $A \rightarrow B_1 \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow B_3 \rightarrow A \rightarrow B_4 \rightarrow A \rightarrow B_5 \rightarrow A \rightarrow D$, while the recursive search route would be $A \rightarrow B_1 \rightarrow B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_5 \rightarrow D \rightarrow B_5 \rightarrow B_4 \rightarrow B_3 \rightarrow B_2 \rightarrow B_1 \rightarrow A$. A P2P protocol could use either one of them. For instance, Kademlia employs the iterative search algorithm, and the Nugache P2P botnet has implemented the recursive routing. Although the routes are different, our analysis applies to both of the search algorithms. This is because, in our analysis, what we care about is the number of distinct nodes that a search message would go through besides node A and node D within a target zone; this number depends on the length of the path $A \rightarrow B_1 \rightarrow B_2 \rightarrow B_3 \rightarrow B_4 \rightarrow B_5 \rightarrow D$, and in both search cases this length is the same.

In addition, as we mentioned before, the ideas of index poisoning and Sybil were first introduced in legitimate P2P networks, and passive monitoring can also be deployed in

current P2P file sharing networks. When P2P botnets use the same P2P protocols, these techniques can be leveraged to fight against these botnets as well. Therefore, our analysis of these three techniques is applicable to not only P2P botnets, but also to legitimate P2P systems. Moreover, in our analysis, we mainly talked about P2P botnets utilizing Kademlia for command and control; however, index poisoning defense and Sybil defense techniques are also valid for P2P botnets that rely on P2P networks for other communication, such as Storm botnet, which utilizes a P2P network to help bots join its hierarchical multi-tier command and control network. Therefore, our analysis is valid for general P2P botnets, no matter whether they use P2P networks for command dissemination, or for other communications.

5.8 Summary

This chapter focused on P2P botnet countermeasures, which include three main directions: detection, monitoring and mitigation. Specifically, we studied two mitigation approaches — index poisoning defense and Sybil defense, and one monitoring approach — passive monitoring. They all are against Kademlia-based P2P botnets. Index poisoning defense and Sybil defense share a similar idea, which is they both try to disrupt the command and control channel by manipulating the command lookup path, thus bots cannot successfully get commands. As for passive monitoring, it is usually carried out by Sybil nodes added by defenders in a botnet, and they passively wait for bots to contact them. Passive monitoring has the advantage of being stealthy. We provided accurate mathematical analysis on the effectiveness of these three techniques, which give guidance to defenders when deploying them.

CHAPTER 6: HONEYPOT-AWARE BOTNETS

6.1 Introduction

In recent years, honeypot techniques have become popular, and security researchers have generated many successful honeypot-based attack analysis and detection systems, such as those we reviewed in Chapter 2. Especially, more people start leveraging honeypot techniques to monitor and mitigate botnets. Because of this, attackers constructing and maintaining botnets will sooner or later try to find ways to avoid honeypot traps.

Therefore, in this chapter, we present how botmasters might attempt to remove honeypot traps when constructing and maintaining their botnets. This knowledge is useful for security professionals to be better prepared for more advanced botnet attacks in the near future. Unlike hardware or software specific honeypot detection methods [102, 44, 21], the honeypot detection methodology presented here is based on a general principle that is hardware and software independent: security defenders who set up honeypots have liability constraint such that they cannot allow their honeypots to send out real attacks to cause damage to others, while botmasters do not need to follow this constraint. As laws are developed to combat cybercrime in the coming years, security experts deploying honeypots will probably incur more liability constraint than they have today, because they knowingly allow their honeypots to be compromised by attackers. If they fail to perform due diligence by securing their honeypot from damaging other machines, they will be considered negligent.

We systematically study honeypot detection that could be deployed by attackers based on the above general principle. Although Lance Spitzner [104] and Richard Salgado [101] addressed the basic potential legal issues of honeypots, their discussion was general and didn't provide details of what attackers might exploit the legal issues and how to deal with such exploits.

Based on this principle, we present a novel honeypot detection approach, which is simple but effective. Botmasters could command their botnets to actively send out malicious traffic (or *counterfeit* malicious traffic) to one or several other compromised computers. These computers behave as "sensors". Botmasters can then determine whether a bot is actually a honeypot or a verified vulnerable victim machine based on whether or not the sensors observe the complete and correct attack traffic transmitted from this bot. Simulation experiments show that current standard honeypot and honeynet programs are vulnerable to the proposed attack.

The above honeypot detection approach will also falsely treat some normal computers as honeypots: these computers are subject to security egress filtering such that their outgoing malicious traffic are blocked. This false positive in honeypot detection, however, does not matter to botmasters. If a bot computer cannot send out malicious traffic, it is better be removed from a botnet, no matter whether it is a honeypot or a well managed normal computer.

To detect a hijacked C&C server in a centralized botnet, botmasters can issue a test command via the server under inspection that causes botnet members to send trivial traffic to the botmasters' "sensors". The hijacked server can then easily be detected if the command is not carried out or is not carried out correctly. In addition, botmasters can detect C&C servers hijacked via DNS redirection [47] by checking whether the IP addresses resolved by DNS queries match the real IP addresses of their servers.

Compared with the currently popular centralized botnets, a P2P botnet is much harder for the security community to monitor and eliminate. Here we present a simple but effective P2P botnet construction technique via a novel "*two-stage reconnaissance*" Internet worm attack, which is also capable of detecting and removing infected honeypots during the worm propagation stage.

6.2 Honeypot Detection in Centralized Botnets

6.2.1 Centralized Botnets

Most botnets currently known in the Internet are controlled by botmasters via a hierarchical network structure, which is highly centralized. Fig. 1.1(a) illustrates the basic network structure of a typical botnet (for simplicity, we only show a botnet with two C&C servers). The bots frequently attempt to connect with one or several C&C servers to retrieve commands from the botnet attacker for further actions. These commands are usually issued from another compromised computer (to hide botmaster's real identity) to all C&C servers. To prevent defenders from shutting down the command and control channel, botmasters usually use multiple redundant servers in their botnets.

To set up C&C servers flexibly, botmasters usually hard-code the servers' domain names rather than their IP addresses in all bots [47]. Botmasters also try to keep their C&C servers mobile by using dynamic DNS (DDNS) [120], a resolution service that facilitates frequent updates and changes in machine location. Each time a C&C server is detected and shut down by its user, botmasters can simply create another C&C server on a new compromised machine and update the DDNS entry to point to the new one.

In the rest of Section 6.2, we discuss how botmasters can thwart the two botnet trapping techniques mentioned in Section 2.3, respectively.

6.2.2 Detection of Honeypot Bots

First, we introduce a method to detect honeypots that are infected and acting as bots in a botnet. The general principle is to have an infected computer send out certain malicious or "counterfeit" malicious traffic to one or several remote computers that are actually controlled by the botmaster. These remote computers behave as "sensors" for the botmaster. If the sensors receive the "complete" and "correct" traffic from the infected host, then the host is

considered "trusted" and is treated as a normal bot instead of a honeypot. Since honeypot administrators do not know which remote computers contacted are the botmaster's sensors and which ones might be innocent computers, they cannot defend against this honeypot detection technique without incurring the risk of attacking innocent computers.

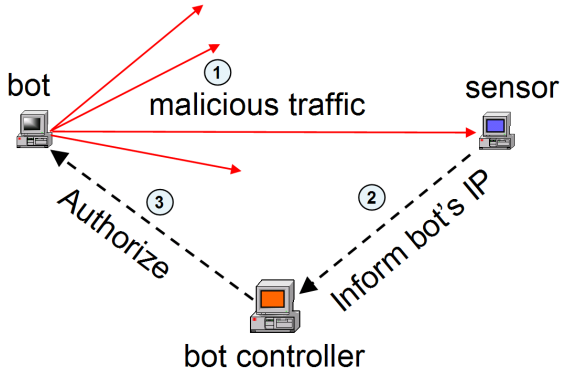


Figure 6.1: Illustration of the procedure of detecting honeypot bots in a centralized botnet

This honeypot detection procedure is illustrated in Fig. 6.1. A newly infected computer cannot join a botnet before it is verified. This potential bot machine must first send out malicious traffic to many targets, including the botmaster's secret sensor (unknown to the newly infected machine). When the botmaster's sensor receives the traffic and verifies the correctness of the traffic (ensuring that it was not modified by a honeypot), the sensor informs the bot controller (e.g. the botmaster or a C&C server) of the bot's IP address. The bot controller then authorizes the checked bot so that the bot can join the botnet. To prevent the possibility of a single point of failure, a botmaster could set up multiple sensors for this test.

This honeypot detection procedure can be performed on a newly infected computer before it is allowed to join a botnet. Such a botnet has a built-in authorization mechanism. The botmaster (or the C&C server) uploads the authorization key to the host and allows it to join the botnet only after the host passes the honeypot detection test. In addition, botmasters may perform the honeypot detection periodically on botnets to discover additional honeypot

bots. This could be done whenever botmasters renew their bots' authorization keys or encryption keys, or update the botnet software.

This honeypot detection scheme relies on the report of sensors deployed by botmasters. Therefore, botmasters must first ensure that sensor machines themselves are not honeypots. This is not hard to be done since only a few sensor machines are needed—botmasters can manually investigate these machines thoroughly beforehand.

Next, we will introduce several illicit activities botmasters might utilize to detect honeypot bots in their centralized botnets.

6.2.2.1 Detection Through Infection

When a computer is compromised and a bot program is installed, some bot programs will continuously try to infect other computers in the Internet. In this case, a honeypot must modify or block the outgoing malicious traffic to prevent infecting others. Based on this liability constraint imposed on honeypot security professionals, a botmaster could let compromised computers send malicious infection traffic to her sensors.

Some honeypots, such as the GenII honeynets [92], have Network Intrusion Prevention System (NIPS) that can modify outbound malicious traffic to disable the exploit. To detect such honeypots, botmasters' sensors need to verify that the traffic sent from bots are not altered (e.g., using MD5 signature).

It is also important that a newly compromised bot does not send malicious traffic to the sensors alone after the initial compromise. It must hide the honeypot checking procedure to prevent defenders from allowing the initial honeypot detection traffic going out. To hide the sensor's identity, a bot could put the sensors' IP addresses at a random point in the IP address list to be scanned. For a bot that infects via email, the sensors' email addresses could be put at a random point in the outgoing email address list. This procedure will delay the newly infected computer's participation in the botnet, but a botmaster would be willing

to incur this slight delay to secure their botnet, because a botnet has long term use to its botmaster.

This honeypot detection technique is difficult for honeypot defenders to deal with. Honeypot defenders cannot block or even modify the outgoing infection traffic. Without accurate binary code analysis, honeypot defenders will not be able to know which target IPs belong to the botmaster's sensors. A botmaster can make the code analysis even harder by obfusticating or encrypting sensors' IP addresses in the code.

6.2.2.2 Detection Through Other Illicit Activities

Based on our general honeypot detection principle, botmasters can have their botnets send out other types of illicit traffic to sensors for honeypot detection. These illicit activities include:

Low rate port scanning. By hiding sensors' IP addresses in the port-scan IP address list, a bot can detect whether or not it is in a honeypot that limits outgoing connection requests. For example, GenII honeynet [92] limits the number of outbound connection rate.

Some normal computers are configured (e.g., installed a firewall, or a worm detection software such as [74]) to limit outgoing connection rate as well. To avoid mislabeling such computers as honeypots, and also to avoid possible detection by users, botmasters should let their bots conduct a very low rate stealthy port-scan for honeypot detection.

Email spamming. A botmaster could also have a bot send out spam email to one or several target email addresses owned by the botmaster. These e-mail addresses behave as the honeypot detection sensors. Outgoing email spam, such as "phishing" email [52], could make honeypot security professionals liable for substantial financial losses if they reach real users.

6.2.3 Detection of Hijacked Command and Control servers

Now we introduce techniques to detect hijacked C&C servers. With the help from Dynamic DNS providers, Dagon *et al.* presented an effective botnet sinkhole that can change the domain name mapping of a detected C&C server to point to a monitoring machine [47]. This way, the monitor receives connection requests from most (if not all) bots in the botnet. Conceptually speaking, the monitor becomes a hijacked C&C server, which is similar to a honeypot in term of functionality.

From a botmaster's perspective, the botnet monitor is very dangerous, because security professionals can learn most of the IP addresses of bots in a botnet — the monitor owners can easily provide a "black-list" of these IP addresses to the security community or potential victims. For this reason, botmasters will do everything they can to eliminate a hijacked C&C server from their botnets. In this section, we present two different techniques that botmasters might use to achieve this goal.

6.2.3.1 Command and Control Server DNS Query Check

When a C&C server is hijacked by the DNS redirection method presented in [47], the IP address of the C&C server returned by DNS query will not be the IP address of the real server. Although bots in a botnet know the domain names instead of the actual IP addresses of the C&C servers, the botnet owner can easily learn all the IP addresses of the botnet's C&C servers, because these computers are compromised by the botmaster and are running the botmaster's bot controlling program.

Therefore, a botmaster can keep an up-to-date DNS mapping table of all C&C servers. Using one compromised computer as a sensor, the botmaster can have this sensor continuously send DNS queries to resolve the name and IP address mapping of all C&C servers in the botnet and then compare the results with the real domain name mapping table. Besides the short time period right after the botmaster changes the C&C server's IP address, this

continuous DNS query procedure is always able to detect whether or not a hijacked C&C server is present in the botnet. If a hijacked server is detected, the botmaster can immediately use other C&C servers to issue a command to update the domain names in all bots, thus obviating further compromise from the hijacked one.

6.2.3.2 Command and Control Server Command Channel Check

The above DNS query check is an effective way to detect DNS redirection of C&C servers. However, it is possible for security defenders to conduct a more stealthy monitoring by actually capturing and monitoring a C&C server. In this case, the DNS query check will not work.

To detect such a physically hijacked C&C server, a botmaster can use the same honeypot detection principle we described before. The botnet owner checks whether or not a C&C server passes the botmaster's commands to bots. The monitor presented in [47] is called "sinkhole" because it does not pass any botmaster's commands to bots. In fact, a hijacked C&C server puts a much more serious liability burden on security defenders than a normal compromised honeypot. If it passes a botmaster's command to bots in a botnet, the defender could potentially be liable for attacks sent out by thousands of computers in the botnet. For this reason, security defenders do not dare to let a hijacked C&C server send out a single command. Even if the command seems harmless from previous experience, it is always possible that a botnet implements its unique command system. In this case, a known trivial command based on previous experience could possibly conduct harmful task such as deleting files on all of the compromised computers, or launching DDoS attacks against risky targets.

Based on this, a botmaster can issue a trivial command to the C&C server under inspection without passing the command to other C&C servers. The trivial command orders a small number of bots to send a specific service request to the botmaster's sensor (e.g., a compromised web server). Bots will not be exposed by this action since they simply send

out some normal service requests. If the sensor does not receive the corresponding service requests, the botmaster knows that the C&C server has been hijacked (or is at least not working as required).

6.2.4 Discussion

Honeypot detection procedure will make an infected computer waiting for a while before it can join a botnet. However, this time delay does not affect the infection speed of computers by a botnet. Because most botnets are used by botmasters as long-time attacking tools, the time delay caused by honeypot detection for infected computers joining a botnet, even as long as several hours or a day, is usually not an issue for botmasters. Therefore, it is not very important for botmasters to consider the trade-off issue between time and accuracy in detecting honeypots.

When deploying a sensor to detect honeypot bots in a botnet, the sensor machine must not be overwhelmed by the testing traffic sent by bots. When conducting honeypot detection, each tested bot only needs to send *one* piece of testing traffic (such as a connection, an email spam, a copy of infection code) to the sensor. Thus a sensor machine is not likely to be overloaded by honeypot detection traffic in most cases, unless bots in a large-size botnet send their test traffic to the sensor at exactly the same time. To prevent such a rare overload event happens in a large-size botnet, the botmaster could command bots to randomly choose their report time within a time period in order to spread the test traffic load.

When using continuous DNS queries to test whether a C&C server is hijacked by defenders or not, the sensor machine sending out these DNS queries might be detected by defenders because of the abnormal DNS queries. Botmasters could prevent this exposure by: (1). slowing down the query frequency; (2). using multiple sensors conducting this query; or (3). sending out DNS queries to many Internet DNS servers.

6.3 Experiment Evaluation

6.3.1 Network System of the Experiments

We conduct experiments to demonstrate the practicability of the proposed honeypot detection methodology by deploying a standard "GenII honeynet" [92]. Fig. 6.2 illustrates the network deployment in our experiments. In this network, there are five computers, and they are functioning as botnet C&C server, botnet sensor (used for honeypot detection), network gateway, honeywall, and honeypot respectively.

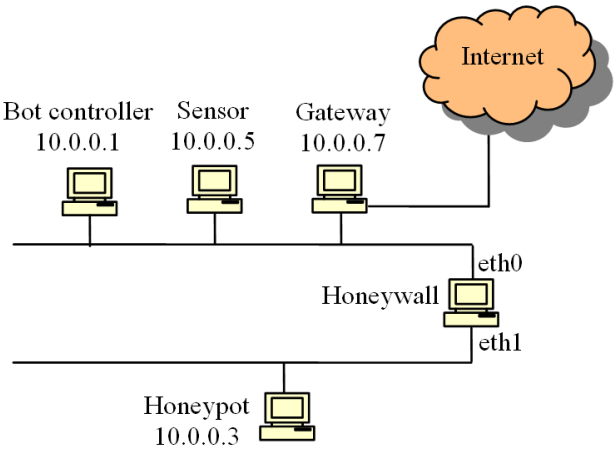


Figure 6.2: Network deployment for honeypot detection experiments

Currently there is no real Internet botnet that has implemented the proposed honeypot detection technique. Therefore, in our experiment we extend the Code Red II worm [20] to generate a basic functioning botnet that incorporates the proposed honeypot detection technique.

The botnet C&C server and the sensor apply the proposed honeypot detection method, cooperating with each other to detect honeypots in the entire botnet. The honeypot is a vulnerable host compromised by the botnet—in experiments, we let the C&C server to send malicious traffic compromising the honeypot. As the botnet continues to spread, the bot program on the compromised honeypot generates a set of random IP addresses, and sends

out packets containing malicious code to hosts with those IP addresses. In addition, the simulated bot program on the honeypot sends out predefined packet to the sensor as well, and this action could happen at any time during the propagation phase.

Honeywall is an important component of a honeynet. It is a gateway device that separates honeypots from the rest of the world. Any traffic going to or from the honeypot must go through the honeywall [92]. We use Honeywall CDROM (version roo) to configure the honeywall, which is working under the bridge mode.

Two data control and capture tools running on honeywall: IPTables and Snort_inline[1]. IPTables behaves as a firewall. It limits the rate of the outbound (from a honeypot to the Internet) traffic, but allows all inbound (from the Internet to a honeypot) traffic. Snort_inline is an intrusion detection system (IDS), monitoring outbound traffic instead of inbound traffic in our experiments. If a packet is not dropped by IPTables, it will be forwarded to Snort_inline, and Snort_inline will analyze it for matches against a user-defined rule set and perform actions such as dropping, modifying, or rejecting the packet, etc [16].

The gateway in Fig. 6.2 is supposed to connect to the Internet, accepting and forwarding any packets targeting all other IP addresses. To prevent malicious traffic going out, in our experiment the gateway does not connect to the Internet and drops all packets going out.

We conduct two types of experiments to show that both data control mechanisms—IPTables and Snort_inline—could cause a honeypot being detected by the proposed attack. In our experiments, instead of using real botnet codes, we emulate botnet infection and bot scanning process.

[1]New version of Snort has integrated the functionality of Snort_inline. What is running on a honeywall is actually Snort. But we still use Snort_inline in the dissertation in order to differentiate Snort_inline from other functionalities of Snort

6.3.2 Honeypot Detection—IPTables

In this experiment, we evaluate the honeypot detection due to the usage of IPTables on honeywall. In our experiment, IPTables is configured to allow 6 outbound TCP connection attempts per hour, and Snort_inline is not running. The abridged log file of IPTables in this experiment is shown below:

```
Oct 11 14:00:29 roo-test kernel: INBOUND TCP: PHYSIN=eth0 PHYSOUT=eth1 SRC=10.0.0.1 DST=10.0.0.3
Oct 11 14:00:29 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=138.230.103.149
Oct 11 14:00:32 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=138.230.103.149
Oct 11 14:00:38 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=138.230.103.149
Oct 11 14:00:50 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=47.217.61.1
Oct 11 14:00:53 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=47.217.61.1
Oct 11 14:00:59 roo-test kernel: OUTBOUND TCP: PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=47.217.61.1
Oct 11 14:01:11 roo-test kernel: Drop TCP > 6 attempts PHYSIN=eth1 PHYSOUT=eth0 SRC=10.0.0.3 DST=10.0.0.5
```

Figure 6.3: Abridged log file of IPTables on the honeywall

The first entry in Fig. 6.3 is an inbound traffic from the C&C server (10.0.0.1) to the honeypot (10.0.0.3). This traffic causes the honeypot to be infected and installed with the bot program. The next 6 entries are all outbound traffic from the honeypot to different destinations, which implies the honeypot has been infected and starts to propagate by sending out probing traffic to randomly generated IP addresses. Only two IP addresses have been scanned because MS Windows will try to connect to each IP address 3 times. After these 6 outbound connection attempts, the IPTables has reached its blocking threshold. The last entry shows that, the outbound packet, which goes to the botnet sensor (10.0.0.5), is dropped by IPTables because the limit of outbound traffic has been reached. In this experiment, the botnet sensor is able to detect the honeypot due to the blocked traffic that originally targets the sensor.

If the packet sending to the botnet sensor is among the first 6 TCP connections, our experiment shows that the packet will successfully arrive the botnet sensor, which could cause the honeypot undetected by the botnet. Therefore, botmasters would mostly likely make their bots connecting to sensors after sending out a number of scans.

6.3.3 Honeypot Detection—Snort_inline

In this experiment, the Snort_inline is activated. One primary function of Snort_inline in honeywall is to detect outgoing malicious traffic, and then either drop it or modify it to be unmalicious. In this experiment, Snort_inline contains the rule shown in Fig. 6.4 to change a *Code Red II* [20] packet to become unmalicious.

```
alert tcp $HONEYNET any <> $EXTERNAL_NET 80 (msg: " Code
Red II detected, modifying its Initial Infection Code "; content:"GET
/default.ida? XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XX%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801
%u9090%u6858%ucbd3%u7801%u9090%u9090%u8190%u00c3%u
0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0";
replace:"[replace with a same length of text]";)
```

Figure 6.4: Snort_inline rule to modify Code Red II outgoing packets

In the experiment, the honeypot sends out a packet containing the Code Red II worm to the sensor, Snort_inline captures the packet and modifies its content, as shown in Fig. 6.4, to a same length of unharmful text. Fig. 6.5 shows the related entry in Snort_inline's log file. Even though the sensor (10.0.0.5) can receive this packet, the botnet still detects the honeypot because the content of the required packet has been changed.

```
[**] Code Red II detected, modifying its Initial Infection Code [**]
10/17-10:12:49.453705 10.0.0.3:4719 -> 10.0.0.5:80
TCP TTL:128 TOS:0x0 ID:64238 IpLen:20 DgmLen:428 DF
***AP*** Seq: 0x449CF7EC    Ack: 0x8B56705E
Win: 0xFFFF    TcpLen: 20
```

Figure 6.5: Related Snort_inline log entry on honeywall

6.4 Honeypot Detection in Peer-to-Peer Botnets

Most current botnets in the Internet use the hierarchical structure (or the centralized topology discussed in [43]) introduced in the previous Section 6.2.1. To increase the availability of the C&C channels in a centralized botnet, a botmaster has to increase the number of C&C servers in the botnet. This will increase the financial cost of maintaining the botnet, since the botmaster will need to purchase more Dynamic DNS domain names. In addition, the botnet is susceptible to C&C server hijacking, which exposes the identity of the entire botnet to security professionals, as was illustrated in [47].

To botmasters, changing a botnet's control architecture to be peer-to-peer is a natural way to make a botnet harder to be shut down by defenders. In recent years, botmasters have tested and implemented different kinds of preliminary P2P botnets such as Slapper [28], Sinit [15], Phatbot [12] and Nugache [77]. Some researchers have studied P2P botnet designs [121,124]. Therefore, we believe more P2P botnets will be created in the near future.

Botmasters will need to come up with a new honeypot detection technique for a P2P botnet. In a P2P botnet, each bot contains a list of IP addresses of other bots that it can connect with, which is called "peer list" [124]. Because there are no central authorities to provide authentication in a P2P botnet, each bot must make its own decision, or collaborate with its peers, to decide whether its hosted machine is a honeypot or not. In this dissertation, we present a simple but effective advanced worm, called "*two-stage reconnaissance worm*", that can be used to distributively detect honeypots as it propagates.

6.4.1 Two-Stage Reconnaissance Worm

A two-stage reconnaissance worm is designed to have two parts: the first part compromises a vulnerable computer and then decides whether this newly infected machine is a honeypot or not; the second part contains the major payload and also the authorization

component allowing the infected host to join the constructed P2P botnet. Due to the different roles in a worm propagation, we call the first part the "*spearhead*", the second part the "*main-force*" of the worm.

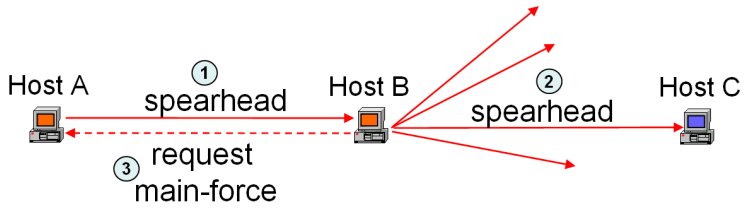


Figure 6.6: Illustration of the propagation procedure of a two-stage reconnaissance worm

A simple way to verify whether a newly compromised host is a honeypot or not is to check whether or not the worm on it can infect other hosts in the Internet. Fig. 6.6 illustrates the propagation procedure of a two-stage reconnaissance worm in infecting host B and checking whether it is a honeypot or not. First, the vulnerable host B is infected by the spearhead of the worm, which contains the exploiting code and the peer list. Second, the spearhead on host B keeps scanning the Internet to find targets (such as host C) to infect with the spearhead code. Third, after the spearhead on host B successfully compromises m hosts (include both vulnerable and already-infected ones), it tries to download the main-force of the worm from any host in its peer list that has the main-force component. The main-force code lets the worm join the constructed botnet via the authorization key contained in the main-force (e.g., the authorization key could be a private public key).

By deploying such a two-stage reconnaissance worm, the botnet is constructed with a certain time delay as the worm spreads. This means that some infected hosts will not be able to join the botnet, since they could be cleaned before the main-force is downloaded. However, this does not affect the botnet, since it makes no difference to the botmaster whether or not the botnet contains bots that will be quickly removed by security defenders.

In fact, it is not a new idea to spread a worm in two stages. Blaster worm and Sasser worm used a basic FTP service to transfer the main code of the worm after compromising

a remote vulnerable host [37]. The two-stage reconnaissance worm presented here can be treated as an advanced two-stage worm by adding the honeypot detection functionality into the first-stage exploit code.

The reconnaissance worm described above needs a separate procedure (Procedure 3 as shown in Fig. 6.6) to obtain the complete bot code. This could be a problem for a botnet since the original Host A might be unaccessible from others, or Host A has changed its IP address when Host Be tries to get the main-force worm code. To deal with this issue, the worm could combine the main-force code together with the spearhead code, but first deactivate and possibly encrypt the main-force code at the beginning. After the spearhead code verifies that a hosted machine is not honeypot, it will unpack and execute the main-force code. One drawback of this approach is that honeypot defenders can easily obtain the main-force code even when their honeypots are not able to join the botnet.

6.4.2 An Advanced Two-Stage Reconnaissance Worm in Response to "Double Honeypot" Defense

Tang and Chen [117] presented a "double-honeypot" system where all the outgoing traffic from the first honeypot is redirected to a dual honeypot. If the dual honeypot is set up to emulate a remote vulnerable host, then the dual honeypot can fool the above two-stage reconnaissance worm into believing that the first honeypot is a real infected host.

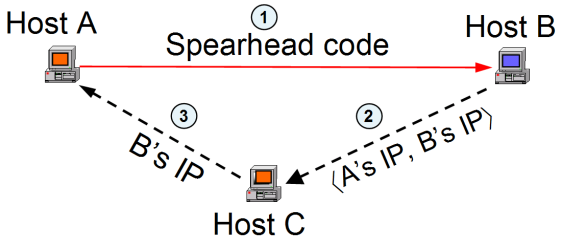


Figure 6.7: The procedure in counterattacking dual-honeypot defense by an advanced two-stage reconnaissance worm (host A is the bot under inspection; host C is in A's peer list that has the main-force code.)

This vulnerability of the two-stage reconnaissance worm is due to: (1). a spearhead makes the decision by itself whether a remote host is infected or not; and (2). a dual honeypot can emulate any outside remote host with arbitrary IP address. To detect such a dual-honeypot defense system, botmasters can design an even more advanced two-stage reconnaissance worm that propagates as following (illustrated in Fig. 6.7):

- When an infected host A finds and infects a target host B, it records host B's IP address. Host A continues finding and infecting others in this way. Host A has a peer list for connecting neighboring bots. The peer list can be constructed according to the P2P botnet designs presented in [121, 124].

- Host B sets up a TCP connection with every host in host A's peer list, telling them the tuple ⟨A's IP, B's IP⟩, which means "host A has sent the correct exploiting code to host B". Suppose host C is one of the hosts in A's peer list. At the time when it receives the tuple ⟨A's IP, B's IP⟩, host C may or may not be a fully-fledged member of the botnet. Host C records this tuple if the incoming TCP connection is really from the claimed host B's IP address.

- After host C obtains the main-force of the worm (host C passes the honeypot detection test earlier than host A), it informs host A of host B's IP address, digitally signed by the private authorization key. This report can be done only by hosts having the main-force code because the authorization key is in the main-force code.

- If host A finds B's IP address in its recorded infection IP list, it knows that it has infected a real host. After host A finds out that it has successfully infected m real hosts, the honeypot detection procedure is over. And then host A tries to download (and execute)the main-force code from any host in its peer list that has the complete code.

This reconnaissance worm will not be fooled by a dual-honeypot system because:

(1) The spearhead in host A chooses IP addresses to scan by itself, thus the real IP address of a dual honeypot has a negligible probability to actually be scanned by host A without IP address redirection;

(2) When host B informs hosts in host A's peer list of the address tuple ⟨A's IP, B's IP⟩, it cannot cheat about its IP address due to the TCP connection;

(3) Only an infected host that is not a honeypot will have the main-force code, and hence, host A can trust that host C is not a honeypot (without this trusted host C, security defenders could use honeypots for all three hosts in Fig. 6.7 to fool the spearhead in host A).

In summary, the advanced reconnaissance worm works because host B cannot lie about its IP address and host C is trusted.

Security defenders in a local network could use a honeynet to cover a large number of local IP addresses. To prevent the spearhead in host A from actually scanning and infecting a local IP address occupied by a honeypot (especially if the worm deploys the "local preference" scans [134]), the worm can conduct the infection report shown in Fig. 6.7 for global infection only, i.e., host B is required to be far away from host A.

6.4.3 Modeling and Analysis of Honeypot Detection Time Delay

As described above, an infected host joins in a botnet only after it has executed the main-force code of the reconnaissance worm. Thus the botnet grows a step behind the propagation of the worm's spearhead. This time delay affects when the botmaster can use the botnet to conduct attacks, or when he/she can upgrade the botnet code. Thus botmasters may be interested in knowing this time delay. In addition, the time delay also affects the attack strength by a new-born botnet (some compromised machines have not joined in the botnet yet due to the honeypot detection procedure), thus security defenders may also be interested

in knowing the delay time. In this section, we study the time delay caused by honeypot detection procedure. We present an analytical model for modeling the growth of a botnet as the two-stage reconnaissance worm spreads.

The modeling presented here tries to show that a two-stage worm will not slow down botnet construction, even though it adds a delay. The modeling results, as presented below, show that all infected computers (not including detected honeypots) will join the botnet in the end, and the machines will join the botnet shortly after the initial infection.

The spearhead of a two-stage reconnaissance worm propagates in way similar to that of a traditional worm, thus it can be modeled by the popular epidemic model as used in [87, 106, 133], etc. Since worm modeling is not the focus of this dissertation, we present a simple model, where the two-stage reconnaissance worm uniformly scans the IP space. Papers such as [71, 134] have presented modeling of local preference scanning, bandwidth-limited spread, and other worm scanning strategies. The model presented here can be extended based on the models in those papers for various non-uniform scanning strategies.

Let $I(t)$ denote the total number of infected hosts at time t — whether a host is infected only by the spearhead or by the full worm; $\bar{I}(t)$ denotes the number of infected hosts that have joined in the botnet by time t, i.e., they have the main-force of the worm. The propagation of the spearhead can be modeled as [106, 133, 134]:

$$\frac{dI(t)}{dt} = \frac{\eta}{\Omega} I(t)[N - I(t)] \qquad \text{(Eq. 6.1)}$$

where N is the total vulnerable population, η is the worm's average scan rate per infected host, Ω is the size of the IP space scanned by the worm.

First, we derive the propagation model of $\bar{I}(t)$ via "infinitesimal analysis" for the two-stage reconnaissance worm with $m = 1$, i.e., a spearhead-infected host downloads the main-force right after it sends out the spearhead and compromises another host. At time t, there are $I(t)$ infected hosts, among them $[I(t) - \bar{I}(t)]$ are infected only by the spearhead — they

have not infected others yet. At the next small time interval δ, each spearhead-only infected host will have the probability $p = \eta\delta N/\Omega$ to infect another host since there are N targets to infect (a target host that has already been infected still counts). Therefore, on average $[I(t) - \bar{I}(t)]p$ spearhead-only infected hosts will infect others and download the main-force of the worm during the small time interval δ. Thus we have,

$$\bar{I}(t+\delta) - \bar{I}(t) = [I(t) - \bar{I}(t)] \cdot p = \frac{\eta}{\Omega}[I(t) - \bar{I}(t)]N \cdot \delta \qquad \text{(Eq. 6.2)}$$

Taking $\delta \to 0$ yields the botnet growth model ($m = 1$):

$$\begin{aligned} \frac{d\bar{I}(t)}{dt} &= \frac{\eta}{\Omega}[I(t) - \bar{I}(t)]N \qquad \text{(Eq. 6.3)} \\ \frac{dI(t)}{dt} &= \frac{\eta}{\Omega}I(t)[N - I(t)] \end{aligned}$$

For a general two-stage reconnaissance worm that has $m > 1$, we can derive the botnet growth model in the similar way. For example, if $m = 2$, then we need to add an intermediate variable $I_1(t)$ to represent the number of spearhead-only infected hosts at time t — each of them has infected exactly one host at time t. Using the similar infinitesimal analysis as illustrated above, we can derive the botnet growth model ($m = 2$):

$$\begin{aligned} \frac{d\bar{I}(t)}{dt} &= \frac{\eta}{\Omega}I_1(t)N \\ \frac{dI_1(t)}{dt} &= \frac{\eta}{\Omega}[I(t) - I_1(t) - \bar{I}(t)]N - \frac{d\bar{I}(t)}{dt} \qquad \text{(Eq. 6.4)} \\ \frac{dI(t)}{dt} &= \frac{\eta}{\Omega}I(t)[N - I(t)] \end{aligned}$$

The above two models assume that the spearhead in host A can download and execute the main-force immediately after it infects m target hosts, which means we assume that at least one of the hosts in A's peer list contains the main-force when A wants to download

the main-force. If the size of the peer list is not too small, this assumption is accurate for modeling purposes.

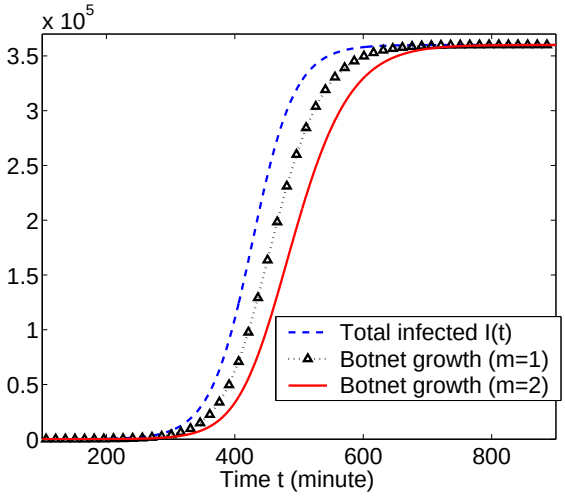


Figure 6.8: Worm propagation and the constructed botnet growth

We use Matlab Simulink [9] to derive the numerical solutions of model (Eq. 6.3) and model (Eq. 6.4). We use the Code Red worm parameters [136], $N = 360,000$, $\eta = 358/\text{min}$, $\Omega = 2^{32}$ in the calculation and assume one initially infected host. Fig. 6.8 shows the worm propagation and the botnet growth over time. The propagation speed relationship would be similar for any other set of worm parameters. This figure shows that the botnet is constructed with a certain time delay (depends on m) as the worm spreads, but in the end all infected hosts will join the P2P botnet. This shows that the method described could potentially produce a viable and large botnet capable of avoiding current botnet monitoring techniques quite rapidly.

6.5 Defense Against Honeypot-Aware Botnets

In this section, we discuss how to defend against the general honeypot-aware attacks (not just botnets) introduced in previous sections.

6.5.1 Loosen the Restrictions on Honeypot Deployment

The honeypot-aware botnet introduced in this dissertation relies on the basic principle that security professionals have liability constraints, while attackers do not need to obey such constraints. The fundamental counterattack by security professionals, therefore, is to invalidate this principle, or at least loosen the restrictions on honeypot deployment, which means allowing honeypots under security professionals' control to actively communicate with other hosts on the Internet. For example, some national organizations or major security companies could set up limited-scale honeypot-based detection systems that are authorized by legal officials to freely send out malicious traffic.

Of course, the law currently regulating cyberspace security is not mature or defined in many countries; hence, some researchers or security defenders have deployed honeypots that freely send out malicious attacks. However, such honeypot defense practices are negligent and unethical. It will become illegal as the laws regarding cyberspace security and liability gradually mature.

The current popular GenII honeynet [92] has considered preventing attack traffic from being sent, but it does not implement this as strictly as the assumption used in the dissertation. First, it limits outgoing connection rate, thus it is possible that some early honeypot detection traffic could be sent out. Second, it can block or modify only *detected* outgoing malicious traffic, thus unknown malicious packets are possibly being sent out by honeypots. For this reason, the GenII honeynet might be able to avoid the honeypot detection conducted by attackers (as explained in Section 6.3); but at the same time, it could actually infect other computers as well and thus potentially make the honeynet owners liable for the ensuing damage.

6.5.2 Crack a Bot

Of course, no one would like to be responsible for damaging other computers, so most of current research work involving honeypots are conducted in a strictly controlled environment, i.e., either outgoing traffic is prohibited, or the packages sent out have to be 100% harmless.

If a honeypot cannot behavior exactly like a real bot, another direction to evade honeypot detection is to start with cracking a bot – understand its honeypot detection mechanism, and then defeat it.

To crack a bot, people usually first obtain a copy of the bot binary, and use it to infect a vulnerable computer in a controlled environment. Thus they are able to monitor its activities on the system and over the network. Additionally, people might be able to get important information that have been hard coded by analyzing the bot code. These reverse engineering technologies are helpful for deciphering the botnet's communication protocol and structure, decoding the botnet commands, locating important information, such as a list of predetermined peers which are bootstrap or command servers. With the ability of knowing a bot, we propose the following ways to counter honeypot-aware botnets.

6.5.2.1 Target the Honeypot Detection Sensors

As we discussed in Section 6.2, when botmasters deploy sensors to detect honeypots by checking test traffic, they rely on the fact that the identities of sensor machines are secret to honeypot defenders. Usually botmaster would set up those sensors and hard code the list of the sensors in the bot code. When a vulnerable host is compromised, this list will be stored on the machine locally. By comparing the system files before and after the infection, it is not hard to locate the file and decode it. Therefore, if security defenders could quickly figure out the identities of sensors before botmasters change their sensor machines, defenders' honeypots could avoid detection by allowing test traffic going out to those sensors.

6.5.2.2 "Double - Honeypot" Technique

For a P2P botnet, it can also employ the sensor based honeypot detection approach, but in order to avoid introducing a single point of failure, in Section 6.4 we proposed the distributed way to detect a honeypot – a two-stage reconnaissance worm. However we also pointed out that "double-honeypot" idea [117] can compromise the whole process. From Section 6.4.2, we can see that attackers need to take complicated extra steps in order to avoid being fooled by double-honeypot traps. By using dual honeypots, or a distributed honeypot network that can accurately emulate the network traffic coming in from the Internet, security defenders can take proactive roles in deceiving honeypot-aware attacks. For example, security defenders can build a large-scale distributed honeynet to cover many blocks of IP space, and allow all malicious traffic to pass freely within this honeypot virtual network. This defense will be ineffective if attackers use their own sensors to detect honeypots (as introduced in Section 6.2), but we have provided the solution above for it.

6.5.2.3 Clone a "Bot"

Sometimes it might be very difficult to evade honeypot detection, such as if defenders cannot figure the identities of the detection sensors, or when dealing with the advanced two-reconnaissance worm (6.4.2). The solution to this case is that we do not try to counter honeypot detection.

As we all know, the main reason that defenders deploy honeypots is to infiltrate a botnet, such that they monitor the botnet and collect information for later analysis. If a honeypot can be identified by attacker, it will not be allowed to join the botnet, but it does not mean that there is no way to infiltrate and monitor the botnet. We can take a little detour. Suppose we can decode the C&C architecture and communication protocols employed by a botnet, and actually current reverse engineering technologies can help us achieve this, we may be able to develop our own "bot", which behaviors in the similar manner but without

the malicious capabilities, just like a "cloned bot". Take "Walowdac" [111] for instance, it is a clone of the Waledac bot, with the communication features of Waledac but without any harm. A host running Walowdac can join the network of Waledac, and publish its identity to other real bots, because of the lack of validation in Waledac botnet. From this example, we can see if a complicated honeypot detection mechanism was implemented in Waledac botnet, a honeypot could not become a real bot, however, by taking advantage of the limitations of botnet design, it is still possible for use to assist honeypots to do what they are supposed to do: infiltration and monitoring.

6.5.3 Discussion

Internet security attack and defense is an endless war.

For defenders, a very essential factor to win this war is to get to know the adversaries. The more we understand about honeypot detection techniques that attackers may leverage, the better that honeypots can perform in the botnet defense. Moreover, advanced reverse engineering technologies are the key to fast and effective response when facing advanced botnet attacks.

From attackers' perspective, there is a trade-off between detecting honeypots in their botnets and avoiding botnet exposure to security professionals. If a botmaster conducts honeypot-aware test on a botnet frequently, honeypots in the botnet can be detected and removed quickly. But at the same time, the bots in the botnet will generate more outgoing traffic, and hence, they have more chance to be detected and removed by their users or security staff.

In the end, we should emphasize that even if attackers can successfully detect and remove honeypots based on the methodology presented in the dissertation, there is still significant value in honeypot research and deployment. Honeypot is a great tool to detect the infection

vector and the source of Internet attacks. It also provides an easy way in capturing attacking code to facilitate security analysis and signature generation.

6.6 Summary

In this chapter, we studied honeypot detection and proposed a software- and hardware-independent honeypot detection methodology which can be applied to designs of honeypot-aware botnets. This methodology is based on a general principle that is unlike attackers who are commanding their bots to conduct malicious activities, defenders cannot allow controlled computers which have infiltrated in a botnet, e.g. honeypots, to launch attacks or harm other computers on the Internet. Potential honeypot detection techniques following this methodology for both centralized botnets and P2P botnets were presented, and have been shown by experiments to be valid. Additionally, we presented ideas of how to defend against such honeypot-aware botnets.

CHAPTER 7: CONCLUSION AND FUTURE WORK

Nowadays, botnets have become a significant part of Internet. They are the major cause of all kinds of attacks and illicit activities taken place on the Internet. Besides traditional centralized botnets, P2P botnet appearing in recent years have shown to be very dangerous. In this work, we explored P2P botnets from multiple angles. After getting a comprehensive understanding of P2P botnets by dissecting P2P botnets' life cycle, we came up with an advanced hybrid P2P botnet with features that make it more resilient and robust against defenses. Then we specially analyzed three effective P2P botnet countermeasures — index poisoning defense, Sybil defense and passive monitoring. Furthermore, we have seen that since honeypot techniques have been used in botnet defense successfully, attackers probably will design honeypot-aware botnets and invalidate the honeypot-based defense systems. Therefore, we studied honeypot detection as well, to find limitations of current honeypot deployment in security defense and remind researchers of the importance of finding ways to build covert honeypots. Our major contributions are summarized in the following section.

7.1 Summary of Contributions

Peer-to-Peer Botnets

(i) A systematic study on P2P botnets.

- The three stages of a P2P botnet's life cycle — bot member selection, network construction and botnet operation, were discussed in detail.
- As a result of the study of P2P botnets' life cycle, a new classification of P2P botnets was presented.

(ii) An advanced hybrid P2P botnet.

- The features presented in this design make this hybrid P2P botnet robust, easy to be controlled and hard to be defended.
- We presented two new metrics — *connected ratio* and *degree ratio* to evaluate the robustness of a P2P botnet.
- The results of our simulation based experiments matched our analytical study of the botnet robustness, also showed the strong resistance of the proposed botnet against defense.

(iii) P2P botnet countermeasures.

- We discussed a number of countermeasures to defend against P2P botnets. We especially studied index poisoning defense and Sybil defense techniques, and presented mathematical analysis to evaluate their performance.
- One counter-intuitive finding was obtained: if the index poisoning defense is valid (when the botnet adopts existing P2P protocols and relies on file index to disseminate commands), P2P botnets are equally easy (or hard) to defend compared to traditional centralized botnets.
- From attackers' perspective, we proposed a novel and realistic technique that might be deployed by them to counter the index poisoning defense. This method guarantees that command related indices can be generated by and only by botmasters, not by ordinary bots.
- We addressed the effectiveness of passive monitoring of P2P botnets by providing a lower bound for the number of bots that a node controlled by defenders can monitor.
- We developed a Kademlia-based P2P botnet simulator. All the analytical work on P2P botnet countermeasures presented in this dissertation have been shown to be accurate by simulation-based experiments.

Honeypot-Aware Botnets

(i) A hardware- and software- independent honeypot detection methodology.

- We pointed out a general security principle that could be exploited by attackers for honeypot detection.
- For centralized botnets, possible honeypot detection techniques based on the general security principle were presented and illustrated to be effective by experiments.
- In order to detect honeypots in P2P botnets, an advanced two-stage reconnaissance worm was designed. Moreover, we presented an analytical model for the growth of a botnet as the two-stage reconnaissance worm spreads, and the analytical results showed that with the ability of detecting honeypots the botnet can still propagate rapidly.
- Potential defense directions against honeypot-aware botnets were discussed.

7.2 Future Work

Our work presented in this dissertation is not enough to win the battle against the next generation botnet attacks. We can further our research in the following directions.

Analysis Verification

- Our analysis on the effectiveness of three P2P botnet defense approaches — index poisoning defense, Sybil defense and passive monitoring was verified by simulation-based experiments in Section 5. To make our analysis more convincing, instead of generating a P2P botnet randomly, we can emulate a real botnet based on the topology obtained from real botnet data. These data can be captured by active monitoring and we found there are some available at [3].

- In addition, in Section 6.4.3, we analyzed the botnet propagation time delay caused by honeypot detection, presented a model and gave numerical results. In order to verify our analysis, and exhibit the feasibility of a honeypot-aware P2P botnet, we can perform simulation experiments on real machines in a distributed environment, such as PlanetLab [13]. We can still use OverSim as the basis of our botnet implementation. And the SingleHost underlay model provided by OverSim, i.e., each OverSim instance only emulates a single host, which can be connected to other instances over existing networks like the Internet, perfectly fits the real network, like PlanetLab.

Passive Monitoring of P2P Botnets When analyzing the effectiveness of passive monitoring of P2P botnets, we gave a lower bound for the number of bots that an infiltrated node can monitor passively. This lower bound only includes bots who have the monitoring node in their routing tables. But as we know, according to Kademlia protocol, the longer the monitoring node stays in the botnet, the more bots it will know. So we can refine our analysis by providing the number of bots an infiltrated node can monitor in terms of the time it has been in the botnet.

Active Monitoring of P2P Botnets In our work, we did not discuss much about active monitoring, which is an important technique as well. For P2P botnets, people usually perform active monitoring by crawling the real network in practice. However, it is worth to have mathematical analysis on the efficiency, accuracy and coverage of active monitoring, starting from its basic idea. This study can give security professionals some ideas of how to effectively carry out this technique.

P2P Botnets Based on Other Protocols In this dissertation, when discussing defense strategies against P2P botnets, we mainly focused on Kademlia-based P2P botnets. As we discussed, there are botnets implementing other P2P protocols, such as Gnutella. It is also very important to study the countermeasures against these types of botnets.

New Types of Botnet Attacks To hide their botnets, attackers always try to make the botnet C&C communication as stealthy as possible. Now besides the communication approaches we have mentioned in this work, botnets have started using social network as their new communication channel. Studying how this communication protocol works, how to disrupt this communication channel and etc, could be an interesting direction to continue our work.

REFERENCES

[1] A Closer Look at Botnets.
http://www.lavasoft.com/support/spywareeducationcenter/botnets.php.

[2] Bittorrent. http://www.bittorrent.com/.

[3] Caida security research datasets. http://www.caida.org/research/security/#Datasets.

[4] Detecting VMware. http://seclists.org/honeypots/2002/q4/0029.html.

[5] eMule. http://www.emule-project.net/.

[6] Gnutella Protocol Specification.
http://wiki.limewire.org/index.php?title=GDF.

[7] Kraken botnet. http://www.iss.net/threats/Kraken.html.

[8] Lime wire. http://www.limewire.com/.

[9] Mathworks Inc.: Simulink. http://www.mathworks.com/products/simulink.

[10] Napster. http://www.napster.com/.

[11] Overnet. http://www.zeropaid.com/overnet/.

[12] Phatbot trojan analysis. http://www.lurhq.com/phatbot.html.

[13] Planetlab. http://www.planet-lab.org/.

[14] SdDrop. http://www.viruslist.com/en/viruses/encyclopedia?virusid=24282.

[15] Sinit P2P trojan analysis. http://www.lurhq.com/sinit.html.

[16] Snort users manual.
http://www.snort.org/docs/snort_htmanuals/htmanual_260/.

[17] University of Michigan Internet Motion Sensor. http://ims.eecs.umich.edu/.

[18] Vbs.Gnutella. http://www.symantec.com/security_response/writeup.jsp?docid=2000-121813-5230-99.

[19] W32/Gnuman.worm. http://vil.nai.com/vil/content/v_99024.htm.

[20] eEye digital security: CodeRedII worm analysis. http://www.eeye.com/html/Research/Advisories/AL20010804.html, 2001.

[21] Honeyd security advisory 2004-001: Remote detection via simple probe packet. http://www.honeyd.org/adv.2004-01.asc, 2004.

[22] CNET news: Bots slim down to get tough, November 2005. http://news.com.com/2104-7355_3-5956143.html.

[23] Washington Post: The botnet trackers, February 2006. http://www.washingtonpost.com/wp-dyn/content/article/2006/02/16/AR2006021601388.html.

[24] Peer-to-Peer Data Mining, Privacy Issues, and Games. White paper, University of Maryland, April 2007.

[25] MessageLabs Intelligence: Q2/June 2009 "Cutwail's bounce-back; Instant messages can lead to instant malware", June 2009. http://www.messagelabs.com/mlireport/MLIReport_2009.06_June_FINAL.pdf.

[26] K.G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A.D. Keromytis. Detecting targeted attacks using shadow honeypots. In *Proceedings of 14th USENIX Security Symposium*, August 2005.

[27] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. In *ACM Computing Surveys*, December 2004.

[28] I. Arce and E. Levy. An analysis of the slapper worm. *IEEE Security & Privacy Magazine*, Jan.-Feb. 2003.

[29] Paul Bäher, Thorsten Holz, Markus Köter, and Georg Wicherski. Know your enemy: Tracking botnets, 2008.
http://csdl2.computer.org/comp/mags/co/2006/04/r4017.pdf.

[30] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and N. Provos. A hybrid honeypot architecture for scalable network monitoring. Technical Report CSE-TR-499-04, U. Michigan, October 2004.

[31] Michael Bailey, Evan Cooke, Farnam Jahanian, Yunjing Xu, and Manish Karir. A Survey of Botnet Technology and Defenses. In *Proc. of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, March 2009.

[32] Paul Barford and Vinod Yegneswaran. An Inside Look at Botnets. *In Series: Advances in Information Security*, 2006.

[33] Ingmar Baumgart, Bernhard Heep, and Stephan Krause. OverSim: A Flexible Overlay Network Simulation Framework. In *Proc. of the 10th IEEE Global Internet Symposium (GI '07) in conjunction with IEEE INFOCOM '07, Anchorage, AK*, May 2007.

[34] J. Bethencourt, J. Franklin, and Mary Vernon. Mapping Internet sensors with probe response attacks. In *Proceedings of USENIX Security Symposium*, pages 193–208, August 2005.

[35] R. Bhagwan, S. Savage, and G. M. Voelker. Understanding availability. In *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems (IPTPS)*, Feburary 2003.

[36] J. R. Binkley and S. Singh. An algorithm for anomaly-based botnet detection. In *USENIX 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI 06)*, June 2006.

[37] CERT. CERT/CC advisories. http://www.cert.org/advisories/.

[38] Su Chang and Thomas E. Daniels. P2P botnet detection using behavior clustering & statistical tests. In *Proc. of the 2nd ACM workshop on Security and artificial intelligence (AISec '09), Chicago*, November 2009.

[39] Su Chang, Linfeng Zhang, Yong Guan, and Thomas E. Daniels. A Framework for P2P Botnets. In *Proc. of the 2009 International Conference on Communications and Mobile Computing (CMC '09), Kunming, Yunnan, China*, January 2009.

[40] Y. Chen. IRC-based botnet detection on high-speed routers, 2006. ARO/DARPA/DHS Special Workshop on Botnet.

[41] Yuqun Chen, Ramarathnam Venkatesan, Matthew Cary, Ruoming Pang, Saurabh Sinha, and Mariusz H. Jakubowski. Oblivious hashing: A stealthy software integrity verification primitive. In *IH '02: Revised Papers from the 5th International Workshop on Information Hiding*, pages 400–414, London, UK, 2003. Springer-Verlag.

[42] Chia Yuan Chox, Juan Caballeroyx, Chris Grierx, Vern Paxsonzx, and Dawn Song. Insights from the Inside: A View of Botnet Management from Infiltration. In *Proc. of the 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats, San Jose, CA*, April 2010.

[43] Evan Cooke, Farnam Jahanian, and Danny McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *Proc. of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI'05)*, July 2005.

[44] J. Corey. Advanced honey pot identification and exploitation. http://www.phrack.org/fakes/p63/p63-0x09.txt, 2004.

[45] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levin, and H. Owen. Honeystat: Local worm detection using honeypots. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[46] David Dagon, Guofei Gu, Chris Lee, and Wenke Lee. A Taxonomy of Botnet Structures. In *Proc. of the 23rd Annual Computer Security Applications Conference (ACSAC'07)*, December 2007.

[47] David Dagon, Cliff C. Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *Proc. of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, February 2006.

[48] Hauke Damfling. Gnutella web caching system. http://www.gnucleus.org/gwebcache/specs.html.

[49] Carlton R. Davis, José M. Fernandez, Stephen Neville, and John McHugh. Sybil Attacks as a mitigation strategy against the Storm botnet. In *Proc. of the 3rd International Conference on Malicious and Unwanted Software (Malware'08)*, October 2008.

[50] Carlton R. Davis, Stephen Neville, José M. Fernandez, Jean-Marc Robert, and John Mchugh. Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures? In *Proceedings of the 13th European Symposium on Research in Computer Security (ESORICS'08)*, pages 461–480, 2008.

[51] John R. Douceur. The Sybil Attack. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems*, March 2002.

[52] C.E. Drake, J.J. Oliver, and E.J. Koontz. Mailfrontier, Inc. whitepaper: Anatomy of a Phishing email. http://www.ceas.cc/papers-2004/114.pdf, 2004.

[53] Brandon Enright, Geoff Voelker, Stefan Savage, Chris Kanich, and Kirill Levchenko. Storm: When Researchers Collide. In *USENIX ;login: 33(4)*, August 2008.

[54] Loras R. Even. Honey Pot Systems Explained, 2000. http://www.sans.org/resources/idfaq/honeypot3.php.

[55] F.C. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. Technical Report AIB-2005-07, CS Dept. of RWTH Aachen University, April 2005.

[56] C. Gkantsidis, T. Karagiannis, P. Rodriguez, and M. Vojnovic. Planet scale software updates. In *Proceedings of ACM SIGCOMM*, September 2006.

[57] Julian B. Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-Peer Botnets: Overview and Case Study. In *Proc. of the 1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots '07), Cambridge, MA*, April 2007.

[58] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proc. of the 17th USENIX Security Symposium (Security'08)*, 2008.

[59] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proc. of the 16th USENIX Security Symposium (Security'07)*, August 2007.

[60] Guofei Gu, Vinod Yegneswaran, Phillip Porras, Jennifer Stoll, and Wenke Lee. Active Botnet Probing to Identify Obscure Command and Control Channels. In *Proc. of the*

Annual Computer Security Applications Conference (ACSAC'09), Honolulu, Hawaii, December 2009.

[61] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[62] Peter Gutmann. World's Most Powerful Supercomputer Goes Online, 2007. http://seclists.org/fulldisclosure/2007/Aug/520.

[63] Kelly Jackson Higgins. Researchers Infiltrate and 'Pollute' Storm Botnet, 2008. http://www.darkreading.com/security/encryption/showArticle.jhtml?articleID=211201340.

[64] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst W. Biersack, and Felix Freiling. Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm. In *Proc. of the 1st Usenix Workshop on Large-scale Exploits and Emergent Threats (LEET), San Francisco, CA, USA*, 2008.

[65] Honeypot. http://www.honeypots.net/.

[66] Márk Jelasity and Vilmos Bilicki. Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In *Proc. of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09), Boston, MA*, April 2009.

[67] X. Jiang and D. Xu. Collapsar: A VM-based architecture for network attack detention center. In *Proceedings of 13th USENIX Security Symposium*, August 2004.

[68] C. Kalt. Internet Relay Chat: Architecture. Request for Comments: RFC 2810, April 2000.

[69] S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *2nd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2005.

[70] Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, and Stefan Savage. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. In *Proc. of the USENIX Workshop on Large-Scale Exploits and Emergent Threats, San Franciso, CA*, April 2008.

[71] G. Kesidis, I. Hamadeh, and S. Jiwasurat. Coupled Kermack-McKendrick models for randomly scanning and bandwidth-saturating Internet worms. In *Proceedings of 3rd International Workshop on QoS in Multiservice IP Networks (QoS-IP)*, pages 101–109, February 2005.

[72] Oleg Kolesnikov, Dave Dagon, and Wenke Lee. Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic. Technical report, Georgia Tech, 2004-2005.

[73] N. Krawetz. Anti-honeypot technology. *IEEE Security & Privacy Magazine*, 2(1), 2004.

[74] C. Kreibich, A. Warfield, J. Crowcroft, S. Hand, and I. Pratt. Using packet symmetry to curtail malicious traffic. In *Proceedings of the Fourth Workshop on Hot Topics in Networks (HotNets-IV)*, November 2005.

[75] Christian Kreibich, Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. On the Spam Campaign Trail. In *Proc. of the USENIX Workshop on Large-Scale Exploits and Emergent Threats, San Franciso, CA*, April 2008.

[76] Christian Kreibich, Chris Kanich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. Spamcraft: An Inside Look at Spam Campaign Orchestration. In *Proc. of the USENIX Workshop on Large-Scale Exploits and Emergent Threats, Boston, MA*, April 2009.

[77] R. Lemos. Bot software looks to improve peerage, May 2006. http://www.securityfocus.com/news/11390.

[78] Zhichun Li, Anup Goyal, Yan Chen, and Vern Paxson. Automating Analysis of Large-Scale Botnet Probing Events. In *Proc. of ACM Symposium on Information, Computer and Communications Security*, 2009.

[79] Jian Liang, Naoum Naoumov, and Keith W. Ross. The Index Poisoning Attack in P2P File Sharing Systems. In *Proc. of the Infocom, Barcelona*, 2006.

[80] M. Liljenstam and D. Nicol. Comparing passive and active worm defenses. In *Proceedings of the 1st International Conference on Quantitative Evaluation of Systems (QEST)*, September 2004.

[81] Wei Lu, Mahbod Tavallaee, and Ali A. Ghorbani. Automatic discovery of botnet communities on large-scale communication networks. In *Proc. of the 2009 ACM Symposium on Information, Computer and Communications Security (ASIACCS), Sydney, Australia*, March 2009.

[82] E. K. Lua, J. Crowcroft, Marcelo Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(2), 2005.

[83] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proc. of the 1st International Workshop on Peer-to-Peer Systems*, pages 53–65, 2002.

[84] B. McCarty. Botnets: Big and bigger. *IEEE Security & Privacy Magazine*, 1(4), July 2003.

[85] Trend Micro. Taxonomy of Botnet Threats. Novermber 2006.

[86] D. Moore, C. Shannon, G.M. Voelker, and S. Savage. Network telescopes: Technical report. Technical Report TR-2004-04, CAIDA, 2004.

[87] D. Nicol and M. Liljenstam. Models of active worm defenses. In *Proceedings of the IPSI Studenica Conference*, June 2004.

[88] Chris Nunnery, Greg Sinclair, and Brent ByungHoon Kang. Tumbling Down the Rabbit Hole: Exploring the Idiosyncrasies of Botmaster Systems in a Multi-Tier Botnet Infrastructure. In *Proc. of the 3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats, San Jose, CA*, April 2010.

[89] Linda Dailey Paulson. Hackers Strengthen Malicious Botnets by Shrinking Them, 2006. http://csdl2.computer.org/comp/mags/co/2006/04/r4017.pdf.

[90] Andreas Pitsillidis, Kirill Levchenko, Christian Kreibich, Chris Kanich, Geoffrey M. Voelker, Vern Paxson, Nicholas Weaver, and Stefan Savage. Botnet Judo: Fighting Spam with Itself. In *Proc. of the Network and Diestributed System Security Symposium (NDSS), San Diego, CA*, February 2010.

[91] Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. A Multi-perspective Analysis of the Storm (Peacomm) Worm. Technical report, SRI, November 2007.

[92] Honeynet Project. Know your enemy: GenII honeynets. http://www.honeynet.org/papers/gen2, 2005.

[93] N. Provos. A virtual honeypot framework. In *Proceedings of 13th USENIX Security Symposium*, August 2004.

[94] Deb Radcliff. Remote Control Wars, June 2006. http://www.scmagazineus.com/remote-control-wars/article/33494/.

[95] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proc. of the 6th ACM SIGCOMM Conference on Internet Measuremen, Rio de Janeiro, Brazil*, October 2006.

[96] A. Ramachandran and N. Feamster. Understanding the network-level behavior of spammers. In *Proceedings of ACM SIGCOMM*, September 2006.

[97] Anirudh Ramachandran, Nick Feamster, and David Dagon. Revealing Botnet Membership Using DNSBL Counter-Intelligence. In *Proc. of the 2nd USENIX Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, July 2006.

[98] Krishna Ramachandran and Biplab Sikdar. Modeling malware propagation in Gnutella type peer-to-peer networks. In *Proc. of the 20th International Parallel and Distributed Processing Symposium (IPDPS '06), Rhodes Island, Greece*, April 2006.

[99] Eric Rescorla. Introduction to Distributed Hash Tables. IAB Plenary, IETF 65.

[100] Elizabeth Van Ruitenbeek and William H. Sanders. Modeling Peer-to-Peer Botnets. In *Proc. of the 5th International Conference on Quantitative Evaluation of Systems (QEST '08), St Malo, France*, September 2008.

[101] R. Salgado. The legal ramifications of operating a honeypot. *IEEE Magazine on Security and Privacy*, 1(2), March 2005.

[102] K. Seifried. Honeypotting with VMware basics. http://www.seifried.org/security/index.php/Honeypotting_With_VMWare_Basics, 2002.

[103] Victor A. Skormin, José G. Delgado-Frias, Dennis L. McGee, Joseph Giordano, Leonard J. Popyack, Vladimir I. Gorodetski, and Alexander O. Tarakanov. BASIS:

A biological approach to system information security. In *MMM-ACNS '01: Proceedings of the International Workshop on Information Assurance in Computer Networks*, pages 127–142, London, UK, 2001. Springer-Verlag.

[104] L. Spitzner. Honeypots: Are they illegal? http://www.securityfocus.com/infocus/1703, 2003.

[105] Lance Spitzner. Honeypots, 2003. http://www.tracking-hackers.com/papers/honeypots.html.

[106] S. Staniford, V. Paxson, and N.Weaver. How to own the Internet in your spare time. In *Proceedings of USENIX Security Symposium*, pages 149–167, August 2002.

[107] Guenther Starnberger, Christopher Kruegel, and Engin Kirda. Overbot - A botnet protocol based on Kademlia. In *Proc. of the 4th International Conference on Security and Privacy in Communication Networks (SecureComm)*, September 2008.

[108] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. A Global View of KAD. In *Proc. of the ACM Internet Measurement Conf. (IMC), San Diego, USA*, October 2007.

[109] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. Exploiting KAD: possible uses and misuses. 37(5):65–70, October 2007.

[110] Joe Stewart. Inside the Storm: Protocols and Encryption of the Storm Botnet, 2008. http://www.blackhat.com/presentations/bh-usa-08/Stewart/BH_US_08_Stewart_Protocols_of_the_Storm.pdf.

[111] Ben Stock, Jan Goel, Markus Engelberth, and Felix C. Freiling. Walowdac - Analysis of a Peer-to-Peer Botnet. In *Proc. of the European Conference on Computer Network Defense (EC2ND '09)*, November 2009.

[112] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, AC. In *Proc. of the ACM SIGCOMM, San Deigo, CA*, August 2001.

[113] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Chris Kruegel, and Giovanni Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *Proc. of the ACM CCS, Chicago, IL*, November 2010.

[114] Sam Stover, Dave Dittrich, John Hernandez, and Sven Dietrich. Analysis of the Storm and Nugache Trojans: P2P is here. *USENIX ;login:*, 32(6):18–27, December 2007.

[115] T. Strayer. Detecting botnets with tight command and control, 2006. ARO/DARPA/DHS Special Workshop on Botnet.

[116] Daniel Stutzbach and Reza Rejaie. Improving Lookup Performance over a Widely-Deployed DHT. In *Proc. of the IEEE INFOCOM, Barcelona, Spain*, April 2006.

[117] Yong Tang and Shigang Chen. Defending Against Internet Worms: A Signature-Based Approach. In *Proc. of the IEEE INFOCOM'05*, May 2005.

[118] Richard Thommes and Mark Coates. Epidemiological Modelling of Peer-to-Peer Viruses and Pollution. In *Proc. of the IEEE Infocom, Barcelona, Spain*, April 2006.

[119] Ricardo Villamarín-Salomón and José Carlos Brustoloni. Bayesian bot detection based on DNS traffic similarity. In *Proc. of the the 24th Annual ACM Symposium on Applied Computing (SAC '09), Honolulu, Hawaii*, March 2009.

[120] P. Vixie, S. Thomson, Y. Rekhter, and J. Bound. Dynamic Updates in the Domain Name System (DNS UPDATE), April 1979. http://www.ietf.org/rfc/rfc2136.txt.

[121] Ryan Vogt, John Aycock, and Michael Jacobson. Army of Botnets. In *Proc. of the 2007 Network and Distributed System Security Symposium (NDSS)*, Febuary 2007.

[122] M. Vrable, J. Ma, J. chen, D. Moore, E. Vandekieft, A.C. Snoeren, G.M. Voelker, and S. Savage. Scalability, fidelity and containment in the potemkin virtual honeyfarm. In *Proceedings of the ACM Symposium on Operating System Principles (SOSP)*, October 2005.

[123] Peng Wang, James Tyra, Eric Chan-Tin, Tyson Malchow, Denis Foo Kune, Nicholas Hopper, and Yongdae Kim. Attacking the Kad Network. In *Proc. of the 4th international conference on Security and privacy in communication netowrks (SecureComm '08)*, August 2008.

[124] Ping Wang, Sherri Sparks, and Cliff C. Zou. An Advanced Hybrid Peer-to-Peer Botnet. In *Proc. of the 1st USENIX Workshop on Hot Topics in Understanding Botnets (HotBots '07), Cambridge, MA*, April 2007.

[125] Ping Wang, Lei Wu, Baber Aslam, and Cliff C. Zou. A Systematic Study on Peer-to-Peer Botnets. In *Proc. of the International Conference on Computer Communications and Networks (ICCCN '09), San Francisco, CA*, August 2009.

[126] Ping Wang, Lei Wu, Ryan Cunningham, and Cliff C. Zou. Honeypot Detection in Advanced Botnet Attacks. In *International Journal of Information and Computer Security (IJICS), 4(1), 30-51*, 2010.

[127] Peter Wurzinger, Leyla Bilge, Thorsten Holz, Jan Goebel, Christopher Kruegel, and Engin Kirda. Automatically Generating Models for Botnet Detection. In *Proc. of the 14th European Symposium on Research in Computer Security (ESORICS), Saint Malo, France*, September 2009.

[128] Liang Xie and Sencun Zhu. A Feasibility Study on Defending Against Ultra-Fast Topological Worms. In *Proc. of The 7th IEEE International Conference on Peer-to-Peer Computing (P2P'07), Galway, Ireland*, September 2007.

[129] Wei Yu, Philip Coyer Boyer, Sriram Chellappan, and Dong Xuan. Peer-to-Peer System-based Active Worm Attacks: Modeling and Analysis. In *Proc. of the IEEE International Conference on Communications (ICC)*, May 2005.

[130] Lidong Zhou, Lintao Zhang, Frank McSherry, Nicole Immorlica, Manuel Costa, and Steve Chien. A First Look at Peer-to-Peer Worms: Threats and Defenses. In *Proc. of the 4th International Workshop on Peer-To-Peer Systems (IPTPS '05)*, February 2005.

[131] Zhaosheng Zhu, Guohan Lu, Yan Chen, Zhi Judy Fu, Phil Roberts, and Keesook Han. Botnet Research Survey. In *Proc. of the 32nd Annual IEEE International Computer Software and Applications (COMPSAC '08)*, July 2008.

[132] Jianwei Zhuge, Thorsten Holz, Xinhui Han, Jinpeng Guo, , and Wei Zou. Characterizing the IRC-based Botnet Phenomenon. Technical report, Peking University and University of Mannheim, 2007.

[133] C.C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for Internet worms. In *Proceedings of 10th ACM Conference on Computer and Communications Security (CCS'03)*, pages 190–199, October 2003.

[134] C.C. Zou, D. Towsley, and W. Gong. On the performance of Internet worm scanning strategies. *Journal of Performance Evaluation*, 63(7), July 2006.

[135] Cliff C. Zou and Ryan Cunningham. Honeypot-Aware Advanced Botnet Construction and Maintenance. In *Proc. of the International Conference on Dependable Systems and Networks (DSN)*, June 2006.

[136] Cliff C. Zou, Weibo Gong, and Don Towsley. Code Red worm propagation modeling and analysis. In *Proc. of the 9th ACM Conference on Computer and Communication Security (CCS '02), Washington DC*, November 2002.