On Detection of Current and Next-Generation Botnets

by

Yuanyuan Zeng

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2012

Doctoral Committee:

Professor Kang G. Shin, Chair
Professor Atul Prakash
Assistant Professor Qiaozhu Mei
Assistant Research Scientist Michael Donald Bailey

© Yuanyuan Zeng 2012

All Rights Reserved

To my parents and my grandma, who are always there for me.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Professor Kang G. Shin, without whom I would not be here writing this dissertation. During my PhD journey, there have been wonderful times and difficult times. Professor Shin always supported, encouraged, guided, and most importantly, believed in me. He gave me a lot of flexibility to pursue topics whatever interested me and helped me build more confidence in myself. I have been very fortunate and proud to work under his supervision. I also want to thank my committee members, Professor Atul Prakash, Professor Qiaozhu Mei and Dr. Michael Bailey, for reviewing my dissertation and providing constructive comments that help me improve my work.

I would like to acknowledge my collaborators for their constant support and invaluable input to my research. Special thanks to Dr. Xin Hu for collaborating with me throughout all these years. He offered a lot of help and advices on my research, and wonderful friendship. Thanks to Dr. Guanhua Yan at Los Alamos National Laboratory, for being an incredible mentor and a caring friend.

I am grateful to the present and former RTCL members for their accompany and helpful discussions from time to time, especially Matthew Knysz, Zhigang Chen, Katharine Chang, Min-Gyu Cho, Hyoil Kim, Alex Min, Jisoo Yang, Xiaoen Ju and Xinyu Zhang. I feel lucky to have Kai-Yuan Hou sit next to me for the past few years and be a good friend of mine. We shared many laughs both inside and outside office, offered each other help and support constantly. My appreciation also goes to Stephen Reger, the Secretary of RTCL, for helping me with administrative matters over these years.

On a personal note, I cannot say enough to thank my dearest parents and grandma for always being there for me in the ups and downs of my life. I would not have been where I am today without their endless and unconditional love and tremendous support. My dissertation is dedicated to them. I also want to thank my beloved fiancé, Bin Hu, for loving me as who I am. When I worked on this dissertation, he not only offered terrific insights and suggestions that inspired me a lot but also provided tons of emotional support and caring. I could not ask for a better friend and partner.

Finally, I would like to acknowledge the US Office of Naval Research for supporting this research under grants N000140910753 and N000140911042.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

On Detection of Current and Next-Generation Botnets

by

Yuanyuan Zeng

Chair: Kang G. Shin

Botnets are one of the most serious security threats to the Internet and its end users. A botnet consists of compromised computers that are remotely coordinated by a botmaster under a Command and Control (C&C) infrastructure. Driven by financial incentives, botmasters leverage botnets to conduct various cybercrimes such as spamming, phishing, identity theft and Distributed-Denial-of-Service (DDoS) attacks. There are three main challenges facing botnet detection. First, code obfuscation is widely employed by current botnets, so signature-based detection is insufficient. Second, the C&C infrastructure of botnets has evolved rapidly. Any detection solution targeting one botnet instance can hardly keep up with this change. Third, the proliferation of powerful smartphones presents a new platform for future botnets. Defense techniques designed for existing botnets may be outsmarted when botnets invade smartphones.

Recognizing these challenges, this dissertation proposes behavior-based botnet detection solutions at three different levels—the end host, the edge network and the Internet infrastructure—from a small scale to a large scale, and investigates the next-generation botnet targeting smartphones. It (1) addresses the problem of botnet seeding by devising a per-process containment scheme for end-host systems; (2) proposes a hybrid botnet detection framework for edge networks utilizing combined host- and network-level information; (3) explores the structural properties of botnet topologies and measures network components' capabilities of large-scale botnet detection at the Internet infrastructure level; and (4) presents a proof-of-concept mobile botnet employing SMS messages as the C&C and P2P as the topology to facilitate future research on countermeasures against next-generation botnets.

The dissertation makes three primary contributions. First, the detection solutions proposed utilize intrinsic and fundamental behavior of botnets and are immune to malware obfuscation and traffic encryption. Second, the solutions are general enough to identify different types of botnets, not a specific botnet instance. They can also be extended to counter next-generation botnet threats. Third, the detection solutions function at multiple levels to meet various detection needs. They each take a different perspective but are highly complementary to each other, forming an integrated botnet detection framework.

CHAPTER I

Introduction

1.1 The Botnet Threat

A botnet consists of bots, which are computers compromised by malware such as worms, trojan horses or backdoors without user consent or knowledge. The botnet herder or the botmaster remotely controls a botnet via a Command and Control (C&C) infrastructure. As illustrated in Figure 1.1, botnets are usually rented and sold in the underground market by the botmasters for financial gains. They can cooperatively launch various cyber crimes: sending out huge volumes of spam emails, hosting phishing web pages, stealing users' identities and mounting Distributed Denial of Service (DDoS) attacks. Botnets are one of today's most serious security threats to the Internet and its end users. According to a recent Symantec report [28], botnets accounted for 77% of all spam sent out in 2010, which was about 10 billion per day on average. Botnet infections are a global pandemic. Microsoft alone detected and removed 6.5 million bot infections from Windows machines around the world in the 2nd quarter of 2010, and the most infections—2.2 million—were in the U.S. [27].

The huge number of bot infections worldwide and the serious damage they have caused make detecting such a threat a pressing and critical task. Botnet detection has been a major topic in the cyber security community for over half a decade. Numerous solutions have been proposed to defend against the botnet threat. Nevertheless, there

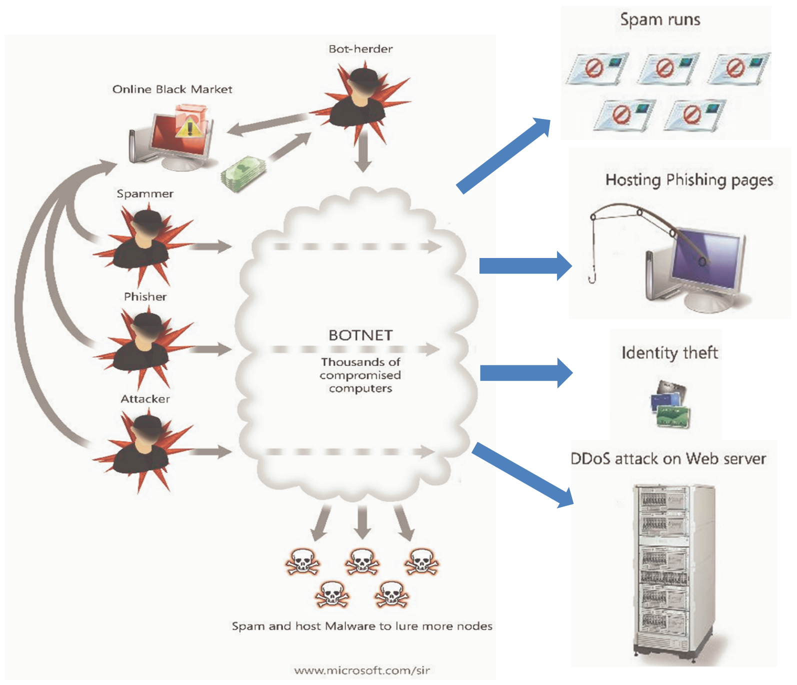


Figure 1.1: The botnet threat

is always arms race between defenders and attackers. State-of-the-art botnets take advantage of multiple techniques and evolve at an unprecedented speed. They present considerable challenges to existing botnet detection approaches.

First, like other types of malware, current botnets commonly employ obfuscation techniques such as polymorphism and metamorphism. Using these techniques, the bot code can mutate without changing the functions or the semantics of its payload. Usually, in the same botnet, bot binaries are different from each other. Since signature-based detection schemes look for specific data patterns in binaries, it is difficult for them to identify all obfuscated bots.

Second, the C&C infrastructure has evolved in recent years. To control a botnet, the botmaster needs a channel to issue commands and coordinate bots. Traditional botnets utilize centralized C&C mechanisms such as Internet Relay Chat (IRC) or HTTP protocols. In the IRC's case, usually, the botmaster takes advantage of an IRC server in a public IRC network by specifying a channel via which bots connect to and listen on to receive commands. HTTP-based botnets are similar to the IRC-based ones. After infection, bots contact a web-based C&C server and notify the server with their system-identifying information via HTTP. This server sends back commands via HTTP responses. IRC- and HTTP-based C&C have been widely used in conventional botnets, but both of them are vulnerable to a single-point-of-failure. That is, once the central IRC or HTTP servers are identified and removed, the entire botnet will be disabled. To be more resilient, attackers have recently utilized decentralized C&C infrastructures such as P2P, where bots do not directly contact any particular servers for commands, but rather retrieve commands from informed peer bots. A well-known example is the Storm botnet [22] which was constructed by the propagation of Storm worm via email spam and is known to be the first malware to seed a botnet in a P2P fashion. Storm utilized Kademlia [61], a decentralized Distributed-Hash-Table (DHT) protocol. The Storm botnet was estimated to run on between 250,000 and 1 million compromised systems in 2007 and was primarily used for sending spam emails. Other noteworthy recent P2P botnets include Waledac [25] and Conficker [6]. In these botnets, a botmaster can join, publish commands and leave at any time at any place. Simply tracking a compromised host can hardly expose the botmaster. Moreover, disabling a certain number of bots does no substantial harm to the botnet as a whole. Thus, botnet detection approaches designed specifically for centralized botnets become less effective for decentralized botnets. Also, given different C&C infrastructures the botmaster can employ, a detection mechanism targeting one C&C instance is not sufficient.

Last but not least, to date, although almost all botnets have been targeting personal computers (PCs), attackers are constantly searching for new opportunities such as new platforms to host botnets. We should be aware that defense techniques targeting state-of-the-art PC-based botnets are likely to be outsmarted when botnets move to a new domain. As the popularity of smartphones such as the iPhone and

Android-based phones grows rapidly, we expect that the botnets are likely to invade smartphones sooner or later. Similar to PC-based botnets, mobile botnets also require three key components: propagation, C&C and topology. Considering the unique features mobile devices have, mobile botnets could take advantage of such features to be more stealthy and resilient to disruption. First, mobile devices can communicate via multiple vectors including SMS/MMS messages, Bluetooth, aside from the conventional IP network. Second, mobile devices move around frequently, and it is generally difficult to find vantage points that can observe all devices' activities. Third, current smartphone users tend to download and share many third-party applications and user-generated contents, but compared to PCs, smartphones have insufficient security protection features, opening doors for cyber crimes. These features together present a good opportunity for future botnets to exploit. Thus, it is important to take the attacker's perspective and think ahead on how to construct the mobile botnets and how to defend against them before they become reality.

1.2 Research Goals

To address these challenges, this dissertation proposes solutions to detect current and next-generation botnets. There are three goals we would like to achieve while designing these solutions.

1. We have observed that bots conduct malicious activities in a coordinated manner so that they demonstrate similar behavior and could distinguish themselves from benign programs or hosts. To successfully combat bots that employ obfuscation techniques, our solutions need to capture invariant properties of botnet behavior without relying on string signatures of binaries or packet payloads.

2. Since state-of-the-art botnets are able to utilize various types of C&C, our solutions should be general enough to detect different botnets instead of targeting

a specific botnet instance and could be extended to counter next-generation botnet threats.

3. Depending on where the detector is deployed, activities that can be captured for use of detection vary from one place to another. For example, in an end-host, fine-grained, OS-level activities such as those in the file system and network stack are all visible, whereas in the Internet infrastructure only traffic flow summaries without packet contents could be recorded. Our solutions must take into consideration the availability of information at different scales (the host, the edge network and the Internet infrastructure) and make the most of the available information to enhance detection accuracy.

1.3 Overview of Existing Approaches

In the literature, numerous approaches have been proposed to detect and mitigate the botnet threat targeting PCs. We briefly overview them based on where the detector is deployed: in the host or in the network.

- **Host-Based Detection:** A bot-infected host behaves similarly to other malware-infected hosts, so general host-based malware detection approaches can be applied. Such approaches either use signature matching or behavior analysis. The latter is of more interest as it can be immune to malware polymorphism and obfuscation. Some behavior-analysis approaches rely on static analysis or examination of executables, such as [34] and [57]. Semantics-aware detection [34] tries to characterize different variations of worms by looking for semantically equivalent instructions in malware variants. In [57], a static analysis is used to identify particular system calls or Internet Explorer API calls that are predefined as malicious. In terms of constructing behavior features, observing system call sequences to identify anomalies is a common approach. Many host-based behavioral approaches [39, 75, 80] focus on profiling the normal behavior by system call sequences and looking for deviations for detection. There are also efforts leveraging runtime analysis. For example, Lee *et al.* [58] collected a sequence of application events at run-time and constructed an opaque object to represent the behavior for further clustering.

- **Network-Based Detection:** Most existing network-based solutions target centralized botnets, i.e., IRC-based and HTTP-based. Gu *et al.* [45] used a network-based anomaly detection to identify centralized botnet C&C channels based on their spatial-temporal correlation. Binkley *et al.* [29] combined an IRC mesh detection component with a TCP scan detection heuristic. Rishi [43] is a detection system that relies on IRC nickname matching. Karasaridis *et al.* [56] proposed the detection of botnet controllers by flow aggregation and feature analysis. Livadas *et al.* [60, 83] utilized supervised machine learning to classify network packets in order to identify the C&C traffic of IRC-based botnets. As P2P botnets emerged, researchers studied the Storm botnet and proposed approaches tailored to P2P-based botnet detection. Holz *et al.* [48] measured the size of the Storm botnet by infiltrating through a crawler, and proposed mitigation strategies that introduce controlled peers to join the network to either separate or pollute the content of the Storm network. Porras *et al.* [70] tried to detect the Storm bot by constructing its dialogue lifecycle model and identifying the traffic that matches this model. All of the above-mentioned approaches only apply to specific types of botnets requiring in-depth understanding of the C&C profiles prior to their detection. There are only a few general approaches. BotMiner [44] is designed for protocol- and structure-independent botnet detection. It clusters similar communication and malicious traffic, and performs cross-plane correlation to identify the hosts that share both patterns. TAMD [96] aims to detect infected hosts within a network by finding those that share common and unusual network communications.

Although most of network-based detection approaches aim to detect bot-infected hosts, there is also a body of research that focuses on identifying botnet-based hosting services, especially fast-flux domains in which the IPs associated with these domains in the DNS (Domain Name System) records change frequently. Such IPs normally belong to bots that serve as proxies or redirection servers, the goal of which is to hide the phishing and malware delivery websites behind the ever-changing network. Holz *et al.* [47] presented an empirical study of fast-flux service networks (FFSNs) and developed metrics to effectively detect FFSNs based on the number of unique A (address) records, NS (name server) records and ASN (AS) records for a specific domain. By continuously mining live data, Nazario and Holz [68] identified over 900 fast-flux domains, and also measured their lifetimes and botnet sizes. Hu *et al.* [49] proposed a system named RB-Seeker that incorporates NetFlow data, spam emails and DNS logs to discover redirection domains.

Note that both of the host-based and network-based approaches have their advantages and disadvantages. The host-based solutions can monitor, capture and analyze fine-grained information in host systems. They are able to know exactly what is going on in the system, so the detection can be targeted and more accurate. However, it is susceptible to compromise by host-resident malware. On the other hand, the network-based approaches are difficult to be subverted but may only have limited view of the botnets, because only network activities are observable in the network.

1.4 Overview of the Dissertation

This dissertation proposes behavior-based botnet detection solutions at three different levels—the end host, the edge network and the Internet infrastructure—from

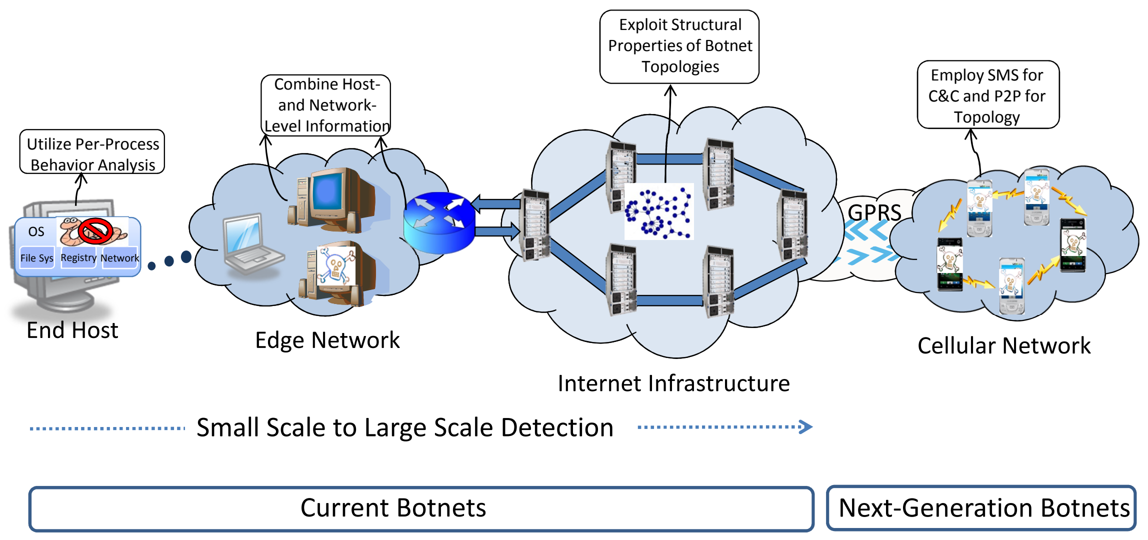

Utilize Per-Process Behavior Analysis
OS
File Sys
Registry
Network
End Host
Combine Host- and Network-Level Information
Edge Network
Exploit Structural Properties of Botnet Topologies
Internet Infrastructure
GPRS
Employ SMS for C&C and P2P for Topology
Cellular Network
Small Scale to Large Scale Detection
Current Botnets
Next-Generation Botnets

Figure 1.2: Overview of the dissertation

a small scale to a large scale, and investigates the next-generation botnet targeting smartphones. To serve multiple detection purposes, one of our solutions is host-based, one is host-network hybrid and the last one is network-based. Figure 1.2 gives an overview of the dissertation. Each piece of work is summarized as follows:

- **Behavior-Based Worm Containment at the End Host:** We start from end hosts because bots are mostly created and spread by network worms from host systems—they propagate by scanning hosts with the same vulnerabilities or by sending emails with malicious attachments or links pointing to nefarious websites. Cutting off such propagation is an important first step in combating the botnet threat. We thus design and implement a behavior-based per-process containment framework on end-host systems. The framework leverages the distinction of OS-wide behavior between benign and malicious processes to generate corresponding traffic-limit policies. The OS-level behavior patterns are monitored and captured at the file system, Registry and network stack. These patterns are further examined by a machine learning algorithm to quantify their suspicion levels. Each suspicion level is finally transformed into a threshold for traffic limiting. Our evaluation results show that the proposed scheme can easily accommodate legitimate applications while effectively containing the propagation of bots and other network worms. This is especially important for mission/service critical systems, because when these systems are infected, completely shutting them down would incur significant loss. The limitation of this host-based solution is that malware can go below our monitoring level and manipulate the information received by our framework. If we can incorporate some external information that is hard to be compromised such as network-level information, it will be more effective.

- **Botnet Detection Using Combined Host- and Network-Level Information in the Edge Network:** Considering that a host-based approach alone may not be reliable enough, we shift our focus to the local network where bots reside in to see if network-level information would be helpful. By studying botnets' behavior, we find that bots within the same botnet usually get the same input from the botmaster and take similar actions thereafter. This coordinated behavior is essential and invariant to all types of botnets irrespective of their underlying C&C structures. Capturing such behavior would facilitate detection, but relying solely on network-level information only has a limited view of botnets' behavior. We believe that incorporating both sources of information will create a synergy. Based on two invariants of botnets—coordination at the network level and malicious behavior at the host level, we design and develop a C&C protocol independent botnet detection framework for edge networks. The evaluation based on real-world traces demonstrates that the framework is able to detect various types (IRC, HTTP and P2P) of botnets with minimal impact on benign hosts, achieving low false alarm rates.

- **Large-Scale Botnet Detection at the Internet Infrastructure:** By monitoring and analyzing fine-grained host and network-level information, the combined-detection framework works well in small-scale networks, such as edge networks. However, current botnet sizes are in the order of hundreds of thousands and bots are distributed over different networks, only detection at the edge is unlikely to harm the functioning of the entire botnet. Moreover, implementing the combined-detection framework is impractical at a large scale. To substantially disrupt a botnet, one must consider detection at a high level—the Internet infrastructure level—to identify as many bots as possible. Following this direction, we construct three types of P2P botnet topologies, investigate the visibility of the botnet overlay traffic at different network components at the Internet infrastructure, measure the effectiveness of detection at such places by exploiting the structural properties of P2P botnets, and evaluate different P2P structures' capabilities of hiding the botnet traffic. This thorough analysis allows us to not only come up with detection strategies from defenders' perspective but also suggest resilient overlay structures from the botnet design's or attackers' viewpoint.

- **The Next-Generation Botnet:** The rapidly-growing popularity of smartphones attracts cyber attackers' attention. Envisioning possible future development of cyber threats targeting smartphones, we devise a proof-of-concept decentralized mobile botnet utilizing SMS messages for all C&C communications and a P2P structure to construct its topology. We simulate two P2P topologies—the structured and the unstructured—for our mobile botnets with 200 nodes and 2000 nodes. We find that the structured topology is a better choice for mobile botnets in terms of message overhead, delay, and load-balancing. As mentioned previously, mobile botnets share some common traits with PC-based botnets, but also have their unique properties. With modifications and extensions, our behavior-based botnet detection solutions aiming at current botnets can be applied to counter this future threat as well.

1.5 Contributions and Organization of the Dissertation

This dissertation mainly makes the following contributions.

1. The botnet detection solutions proposed in this dissertation utilize intrinsic and fundamental behavior of botnets: malicious OS activities at the host level, coordination at the network level and structural topology at the Internet infrastructure level. Without relying on signatures of binaries or packet payloads, these solutions are immune to malware obfuscation and traffic encryption.

2. The detection solutions are general enough to identify different types of botnets, not a specific botnet instance, requiring almost no a priori knowledge of C&C protocol details. They can also be extended to counter future botnet threats. For example, the host-based behavioral detection can be modified to deploy to mobile devices to identify mobile malware. Another example is that the principles of detection at the infrastructure level can be applied to 3G or 4G cellular networks to capture mobile devices whose communication graphs have structural properties.

3. The detection solutions function at multiple levels—the host, the edge network and the Internet infrastructure—from a small scale to a large scale. They each take a different perspective but are highly complementary to each other, forming an integrated botnet detection framework.

The remainder of the dissertation is organized as follows. Chapter II addresses the problem of worm propagation that is used to seed botnets by devising a behavior-based per-process containment scheme on end-host systems. Chapter III proposes a C&C protocol independent framework for botnet detection in edge networks. Using combined host- and network-level information, this framework is able to detect different types of botnets with minimal impact on benign hosts. Chapter IV considers the scalability issue in botnet detection and exploits the structural properties of botnet topologies from a graph perspective at a high level. It focuses on measuring different network components' capabilities for large-scale P2P botnet detection at the Internet infrastructure level. Chapter V presents the design of a next-generation botnet targeting smartphones. The botnet employs SMS messages as C&C and utilizes a P2P topology to be stealthy and resilient. Countermeasures against this threat are also discussed in this chapter. Chapter VI concludes this dissertation.

CHAPTER II

Behavior-Based Worm Containment at the End Host

2.1 Introduction

In recent years, there has been an exponential surge in both the number of network worms and the severity of damage they have inflicted [84]. Fast-spreading worms, such as Blaster (2003), MyDoom (2004), Zotob (2005), Storm (2007), propagated at an unprecedented rate and could infect most vulnerable systems within a short period of time. The intent of a worm has evolved from simply replicating itself to installing malicious payload in the victim systems for collecting confidential information and perpetrating other attacks. Current worms are mostly used to seed botnets, one of the most serious security threats to the Internet and its end users. Worms propagate either through vulnerability scanning or through social engineering schemes such as sending out spam emails with malicious attachments or links pointing to nefarious websites. For example, the Storm worm came out in early 2007; it spread via infected email attachments and once accounted for 8% of all malware infections on Microsoft Windows computers globally. Each compromised machine then merged into the well-known Storm botnet under a decentralized P2P C&C. The Storm botnet remained active for two years, infecting millions of machines to conduct spamming and DDoS attacks. Evidently, to nip the botnet in the bud, alleviating the problem of worm propagation from end host systems is an important first step.

To combat fast-spreading worms, numerous solutions have been proposed to detect and automatically respond to worm outbreaks. A widely-used approach is the signature-based detection, which looks for specific signatures (usually raw byte sequences) in the application executables. The disadvantage of this scheme is that it can only detect previously-known worms and can be evaded even with simple variations thereof. Behavior-based detection has recently received considerable attention due to its capability of identifying new attacks [34, 39, 57, 75, 80, 88]. Most of prior work requires direct analysis of the binaries [34, 57] or system call sequences [39, 75, 80]. Also, the purpose of behavior-based detection is to classify each application as malicious or benign, which may result in high false-alarm rates due to the ambiguity of behavior-matching.

For fast and effective containment of worms, an automatic response is of particular interest because any method that requires human intervention is much slower than the spreading speed of current worms. The detect-and-block approaches could eliminate the human intervention in the loop, but they may not be an option for mission- or service-critical systems such as air traffic control systems, life support systems and servers running critical business services. Obviously, when such systems are infected by worms, immediately taking them offline will incur significant loss to businesses and even pose danger to peoples' lives. Under these scenarios, we would like to contain the malicious network traffic as much as possible and still keep the benign services and applications running until the critical tasks are finished or taken over by other healthy systems. That's why we resort to rate-limiting—an alternative to the detect-and-block approaches. The main idea of rate-limiting is to block the propagation of worms while allowing legitimate traffic to go through, by differentiating traffic patterns between legitimate applications and network worms. Rate-limiting cannot completely block worms, but can significantly slow down the propagation of (especially new) worms, allowing for other countermeasures to kick in.

A key element to worm containment is the selection of a metric based on which the traffic is rate-limited. Previous research results [33, 74, 93] suggest several metrics derived from host-level network activities, such as distinct IP connection ratio, failed connection ratio, the number of connections without DNS queries, etc. Another key aspect of rate-limiting is the use of a threshold beyond which the outgoing traffic is blocked. Most existing containment schemes impose a static threshold rate on the *entire* host, such as several distinct IP connections per second. Sekar *et al.* [76] proposed use of different detection thresholds during different time windows for each host. Rate-limiting on a per-host basis has advantages and drawbacks. The advantages are: (1) the mechanism can be implemented in the network without the trouble of deploying monitors in each host; (2) it is relatively difficult for malware to tamper with the network traffic statistics. The drawbacks are also obvious. First, rate-limiting on the entire host is likely to cause both false-positives and false-negatives. False-positives stem from the coarse-grained rate-limiting policies applied indiscriminately to both normal and malicious processes. Legitimate traffic will therefore be affected significantly during a worm outbreak. It is likely that all legitimate traffic is dropped because the amount of malicious traffic exceeds the threshold, defeating the purpose of rate-limiting. This is undesirable especially when infected mission- or service-critical systems need to keep certain applications or services uninterrupted. False-negatives may result from the evasion of detection by worms that have traffic patterns similar to that of normal applications. This is undesirable either. Second, sometimes it is necessary to scrutinize where and how the worm infection starts and pinpoint which application/process is responsible for that. Only monitoring traffic in the network can hardly provide such information. Fine-grained monitoring and analysis is needed at the host level. This prompts us to consider *per-process* behavior in the worm containment to enhance rate-limiting accuracy.

We propose a per-process-based containment framework. We define behavior at a higher level than others: a sequence of events rather than that of system calls or API calls. Moreover, the behavior profiles of both worms and normal programs are characterized to utilize the notion of anomaly as well as misuse analysis. Our framework considers not only network activities but also a variety of notable behaviors common to network worms, such as creating AutoRun Registry key, overwriting system directories, etc. To compensate for the inaccuracy of behavior analysis and make the best of behavioral information, we use a machine-learning algorithm, instead of making a clear-cut (binary) decision of malice or innocence, to assign a *suspicion level* to each process based on the comprehensive analysis of its behavior. The suspicion level is then transformed into a threshold to rate-limit the process. Since the generation of a suspicion level incorporates many more process-related properties than network activities alone, the containment scheme can make an accurate and flexible decision on how to rate-limit a process, thus lowering false-positive and false-negative rates.

2.1.1 Contributions

Our contributions are three-fold. First, we propose a framework incorporating both behavior analysis and containment for automatic defense against fast-spreading network worms. Our framework differs from others in that, instead of per-host rate-limiting based solely on network activities, it incorporates a comprehensive analysis of processes' behavior and performs customized rate-limiting on each process. This fine-grained monitoring and analysis significantly improve the effectiveness of rate-limiting. Second, we apply a machine-learning classification algorithm to generate a suspicion level for each process and develop a heuristic to find an optimal function that maps each suspicion level to a threshold for rate-limiting. Third, we conduct in-depth analysis and simulation using the traces of real-world worm samples plus

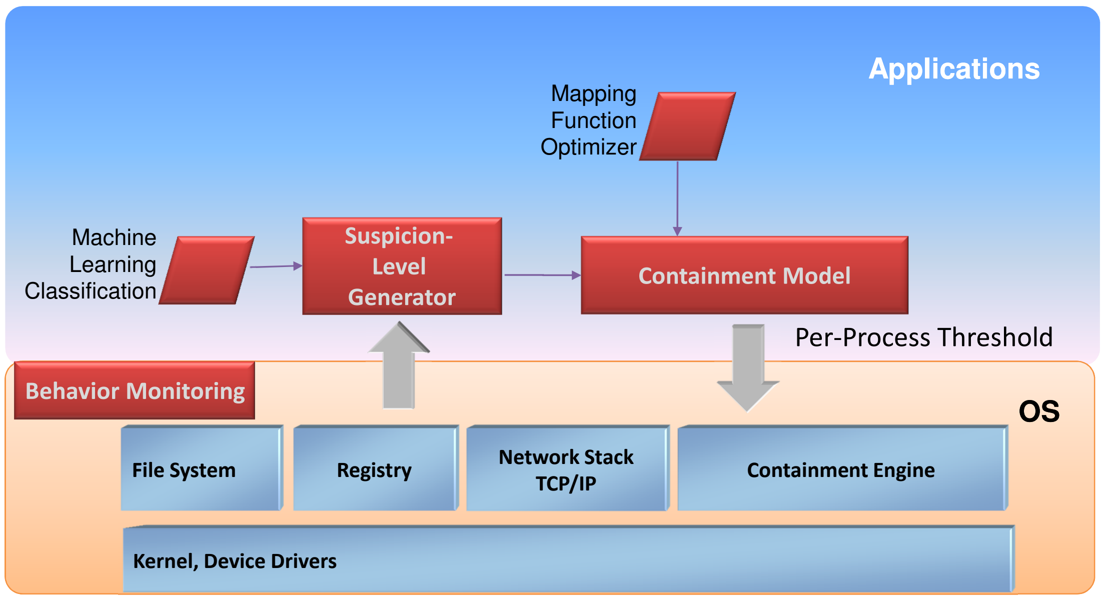


Figure 2.1: System architecture

their variants and normal programs. Our evaluation results show that the proposed scheme can easily accommodate legitimate applications while effectively containing the propagation of network worms. Our fine-grained per-process thresholding can achieve much lower false-positive and false-negative rates than the per-host approach.

2.1.2 Organization

The remainder of the chapter is organized as follows. Section 3.2 provides an overview of our system architecture. Section 2.3 details the process-level behavior analysis. Section 2.4 presents the principles of containment. Implementation and evaluation results are presented in Section 4.4. Section 2.6 discusses the limitations of our work and their solutions. This chapter concludes with Section 2.7.

2.2 System Architecture

Our framework (Figure2.1) primarily consists of two building blocks: behavior analysis and containment. The behavior analysis component includes several system monitors and a suspicion-level generator. Runtime behavior for each process is monitored at the OS level, such as Registry, file system and network stack. Process correlation is tracked as well. The suspicion-level generator assigns a suspicion level to each running process by applying the SVM algorithm based on the analysis of its system-wide activities. The suspicion level links process-level behavior to containment. For containment, the mapping function optimizer generates the most appropriate function of transforming the suspicion level to a containment threshold. Both the suspicion-levels and the mapping function are taken as the input to the containment model which then outputs a customized threshold for each process. In what follows, we will detail each component.

2.3 Behavior Analysis

The first step to combat the propagation of worms is to identify processes conducting malicious activities in a host system. Previous containment techniques confine themselves to network activities, such as high failure rate and the absence of DNS query, in order to identify suspicious traffic. In this paper, we employ behavior-based analysis that focuses on application run-time behavior including Registry, file system and network. By studying contemporary worms' behaviors, we have observed that they do share certain behavior patterns (e.g., creating autorun registry key, scanning random host IPs) that are different from normal applications.

2.3.1 Behavior Signature Specification

We define a *behavior signature* as the description of an application's activities in terms of its resource access patterns. Our goal is to develop a simple yet efficient representation of application behavior that maximally differentiates legitimate applications from worms so that suspicion-level information can be generated to facilitate per-process containment. Note that a single activity—such as network access, a file read or written during a worm's life time—alone may appear harmless, while the combination of these activities may reveal a malicious intent. We thus specify a behavior signature as the aggregation of *suspicious* activities that can potentially be exploited by network worms. To design an efficient specification of worm activities, we need to extract the "common" behavior of network worms, which can be understood better by looking at a few notable examples. From real-world worms and their variants, we found that the worm actions can be grouped into 3 categories, taking place at Registry, file system and network stack, where our behavior monitors are deployed.

Registry : A common target of worms is the AutoRun Key `HKLM\Software \Microsoft\Windows\CurrentVersion\Run`. Most, if not all, worms will add an entry under this key to automatically run themselves when Windows starts up. Examples include Zotob, Win32-Blaster, and W32-Bozori. Some worms also create Registry keys such as `HKEY_CLASSES_ROOT\CLSID\Random_CLSID\ InprocServer32\(Default)` to conceal its backdoor by injecting into other processes. By setting this registry value to the name of the backdoor DLL file, some benign processes (in the above case, explorer.exe) will load the DLL as an extension when the system starts up, so that the backdoor is not visible as a separate process. For example, Mydoom sets this registry key to be "shimgapi.dll"—a backdoor it dropped in the system directory listening on a port between 3127 and 3198.

File System : Once a system is infected, a worm always downloads its payload from the network to the local file system so that it can be activated again when the system reboots. Almost all worms choose the system directory (e.g., `C:\WINDOWS`) as an ideal place to drop themselves, because normal users seldom inspect the system directory and the worm payload is less noticeable among thousands of system files. For instance, Win32-Mydoom creates taskmon.exe and Win32-Bobax drops bleh.exe into the system directory. Both of them change the registry key to make these two files automatically execute. Based on this observation, we closely monitor the create and write accesses in system

Table 2.1: Index of behavior vectors

Index	Signature Description
0	Number of First-Contact Connections
1	DNS-to-Connection Ratio
2	Number of Suspicious Ports
3	Average Packet Length
4	Number of Packets
5	Modify Dll in System Dir
6	Modify EXE in System Dir
7	Modify other files in System Dir
8	Create Dll to System Dir
9	Create EXE to System Dir
10	Create other files to System Dir
11	Create AutoRun key in Registry
12	Set AutoRun key Value in Registry
13	Create DLL injection key in Registry
14	Set DLL injection key Value in Registry

directories.

Network : This category of actions are taken by self-propagating worms, whose goal is to infect as many hosts as possible. For example, Blaster probes 20 hosts at a time using a sequential scanning algorithm with a random starting point. Zotob creates 300 threads to connect to random IP addresses within the B-class network of the infected system. Slammer simply generates UDP packets carrying copies of itself at the host's maximum rate. Thus, intensive network accesses are a good indicator for scanning activities.

Note that none of the above activities is inherently malicious, because they are also performed frequently by many normal applications. However, the combination and accumulation of these activities are essential to the detection of malicious intents with a high degree of confidence, as very few legitimate applications will conduct these activities altogether and intensively. We thus construct a vector of behavior features for each process. we also consider process correlation to defend against sophisticated worms that create multiple processes upon execution, which we will describe later. Each feature in the behavior vector represents one type of application behavior of interest. Table 2.1 summarizes these behavior features that constitute the signature vector. In addition to the behaviors that fall directly into the above three categories, we also take advantage of some auxiliary network features that differentiate worm activities from normal ones. These include DNS-to-connection ratio, suspicious port, and average packet length. The DNS-to-connection ratio is of interest to us because most worms scan *random* IP addresses without DNS queries. The average packet length also provides a hint for suspicious behavior, as worms usually send many identical short packets for both efficiency and fast propagation. Number of suspicious ports records the number of connections initiated by the process to a set of potential vulnerable ports such as 135 and 445. Each behavior feature is associated with a numeric value indicating the number of occurrences of that behavior. The high-level behavior signature for a process is constructed as a vector of all the features, which is then used to determine the suspicion level of the process.

2.3.2 Suspicion-Level Generator

To respond quickly to fast-spreading (especially previously unknown) worms, we build our behavior analysis upon the *Support Vector Machine* (SVM) [55, 87] that learns the behavior models from both normal and malicious behavior signatures. We collect behaviors from normal applications and worms and generate the corresponding behavior vectors as training data. The SVM algorithm maps training data into a higher-dimensional feature space using a kernel function and determines the maximal margin hyperplane to best separate normal data from malicious data. The hyperplane corresponds to the classification rule. Given a test sample, the SVM calculates the distance of the sample from the separating hyperplane with a sign indicating which class (malicious or benign) the test sample belongs to.

Previous research focused on the binary (malicious or benign) classification and the results are likely to be inaccurate because of the learning procedure. To make the best of the learning model, we calibrate the distance score to a posterior classification probability, which determines how likely a test example belongs to a certain class [59]. The posterior probability is then directly translated into the *suspicion level* between 0 and 1 where 0 (1) means benign (malicious). Apparently, the higher the suspicion level, the more likely the process is malicious, and thus, a stricter containment action should apply. The extension from binary classification to a suspicion level facilitates customization of the containment method for each process. Worm traffic is more likely to be strictly rate-limited while legitimate applications will experience a minor traffic-limiting impact.

It is important to note that our suspicion-level generation is not a one-time rating but a periodic check. It can capture all of runtime behaviors of interest and provide a suspicion level for each process during every time window. Thus, a worm that replaces its process ID with a normal program or attaches itself to a normal program is unlikely to affect our decision. For example, if the Internet Explorer (IE) is the target of the worm, its suspicion level will be high as long as it exhibits some bad behaviors, and its traffic will thus be contained. Some legitimate traffic from IE may be affected, but the process-level containment is the finest-grain one can achieve.

2.3.3 Process Correlation

Most worms to-date behave badly on their own, while some sophisticated worms may have multiple processes collaboratively conduct malicious activities. To defend against such a worm whose single processes are not malicious enough to trigger effective rate-limiting, we account for process correlation while building the behavior vectors. We track the inter-process relationships and aggregate the behaviors from correlated processes. The behavior vectors are the same for correlated processes such as a parent process and its children. Accordingly, the whole group of correlated processes is assigned the same suspicion level. By maintaining a white list, we can easily exclude some normal processes with correlation such as services and svchost. Thus, it is not difficult to identify a group of processes behaving maliciously altogether.

2.3.4 Behavior Accumulation

Since the suspicion level is generated every time window, a worm can hide itself by appearing benign for each window but malicious overall. In other words, it may spread suspicious behavior over different time windows or reduce the intensity of malicious activities within a single window, thus decreasing its suspicion level. To deal with such worms, we selectively accumulate the value in each field of the behavior vector. The behavior features worth accumulation are those seldom seen from normal programs, such as creating an autorun key in the Registry or dropping a dll into the system directory, etc. As for some behavior shared by both normal and malicious programs such as outgoing connections, we do not accumulate the value in order not to increase false-positives. The accumulation is straightforward. For example, a worm registers an autorun entry in the registry in window 0 and drops a backdoor in the system directory in window 1. Suppose the behavior vector's first two fields are $\langle autorun\ key, dll\ drop, \ldots\rangle$. The vector in window 1 will be $\langle 1, 1, \ldots\rangle$ instead of $\langle 0, 1, \ldots\rangle$. This way, even if a worm does only one bad thing in each time window to lower its suspicion level, the suspicion level will finally increase as more malicious activities are exhibited. This mechanism also works for a worm spawning multiple processes with each exhibiting malicious activities in different time windows.

As we do not have such a real-world worm sample available, to evaluate the accumulation scheme we simulated a worm in experiment to illustrate the difference of suspicion level between the original and the accumulated scheme. The results are in Section 4.4.

We will next present a model and an algorithm used to transform the suspicion level to an appropriate containment threshold.

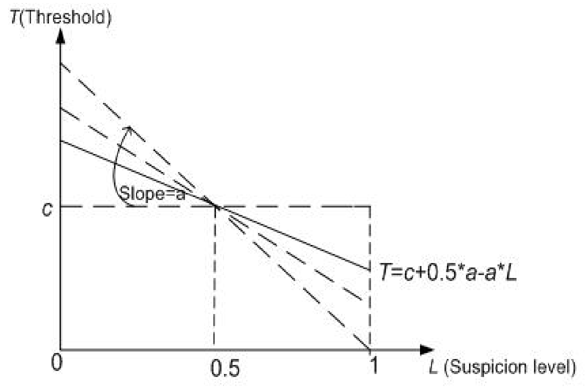


Figure 2.2: Static threshold vs. per-process threshold

2.4 Per-Process Containment

The containment scheme seeks to rate-limit the propagation of network worms while allowing for the operation of normal programs on a host. As mentioned earlier, each process's suspicion level is computed for a time window based on the activities observed during the last time window. This suspicion-level information is then mapped to a threshold. The threshold indicates a connection rate value beyond which the outgoing connections will be blocked. The threshold for each process changes as a process's runtime behavior differs during each time window.

2.4.1 The Mapping Function

The key element in our approach is how to map each suspicion level to a threshold beyond which the process is rate-limited. This mapping function can be of any form but should have a common property; the set of thresholds for processes with higher suspicion levels should be lower and for those with lower suspicion levels should be higher. The rationale behind this is that when the suspicion level for a process is high (low), we would like to block its outgoing traffic as much (little) as possible. The mapping function should therefore be monotonically decreasing within [0,1]. Any type of functions satisfying this property can be used for our purpose. For computation and comparison convenience, linear functions are adopted in our model. Actually, linear functions are found to work well in Section 4.4. Specifically, we choose $t = h(l) = c + 0.5 * a - a * l, a, c \geq 0, l \in [0, 1]$ as the class of mapping functions (Figure 2.2), where t denotes the threshold, l is the suspicion level, and a and c are two design parameters. The smaller the a, the less suspicion-level information is used. When the slope $a \to 0$, it is equivalent to using a static threshold to all processes, ignoring the suspicion level. Parameter c reflects the tolerance to the false-positive rate. Given a, the larger the c, the higher thresholds assigned to all processes. Given the form of the mapping function, it is crucial to choose appropriate values of a and c. The criteria for the efficacy of containment are false-positive and false-negative rates. To calculate the false alarm rates, we develop a probabilistic model by assuming certain properties of normal and malicious processes in order to obtain false-positive and false-negative profiles in terms of a and c. Based on the false alarm profiles, an optimization algorithm is designed to find the appropriate parameters for the mapping function. Our model and algorithm are presented next.

2.4.2 Modeling False Alarms

2.4.2.1 Assumptions

The following assumptions are used in this model.

1. A process is either normal or malicious.

2. The suspicion level across all processes could be treated as a random variable denoted by L_0 (L_1) for normal (malicious) processes, where $L_0, L_1 \in [0, 1]$. $L_0(L_1)$ has cdf $F_0(F_1)$ and pdf $f_0(f_1)$.

3. A mapping function h is from $L(L = L_0$ or $L_1)$ to T (threshold). The threshold for normal (malicious) processes is denoted by T_0 (T_1).

4. First-contact outgoing connection (connection to an address the sender has not recently contacted) rates denoted by R_0 for normal and R_1 for malicious pro-

Table 2.2: Connection-rate statistics

	Average Conn Rate	Standard Deviation
Normal	1.75	1.40
Malicious	6.08	5.00

cesses follow certain distributions: $F_{conn0}(f_{conn0})$ for normal and $F_{conn1}(f_{conn1})$ for malicious processes. Those connections are of interest because malicious programs tend to reach as many hosts as possible while normal programs have the "locality" property in outgoing traffic.

2.4.2.2 Data Analysis

We estimated the distributions of first-contact outgoing connections based on real-world traces. For normal programs, we used attack-free network traffic by tcpdump that lasts 17187 seconds, including 585,000 frames. We have selected all new connections initiated, and filtered out other traffic. The connection rate is defined as the number of connections per second. For malicious programs, due to relatively limited access to the real-world network worms, we collect network activities from 10 types of worms and some of their variants. We set up 3 virtual machines connected via a virtual network as our test-bed to collect the network activity data. The connection rate CDFs are shown in Figure 2.3. We find that 60% of the normal programs' connection rates are around 1/s and 100% of their rates are below 7/s. On the other hand, 50% of malicious programs' connection rates concentrate in the range from 5/s to 8/s. The average and standard deviation across all normal and malicious processes are given in Table 2.2. Clearly, worm's connection rate is, in general, higher than that of normal programs reflecting the fast propagation of network worms.

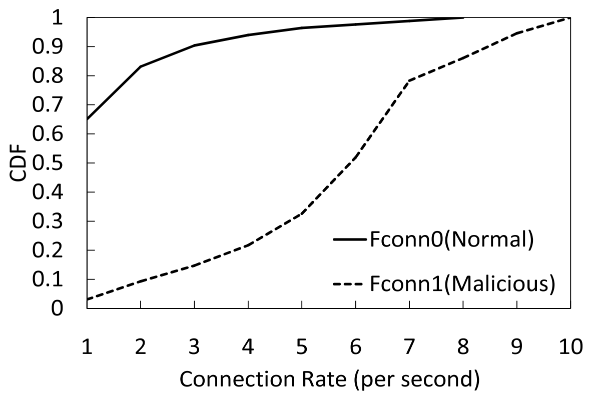


Figure 2.3: Connection-rate CDFs

2.4.2.3 False-Alarm Equations

A false-positive occurs when the connection rate of a normal process exceeds its threshold. A false-negative occurs when the connection rate of a malicious process is below its threshold. If the threshold is static for all processes in a host, it can not effectively contain worms and accommodate normal applications at the same time. Our proposed rate-limiting is to assign each process with a customized threshold, which is much finer-grained. To compare it against static threshold approach in terms of false alarms, we derive the false-alarm equations for both approaches. Those equations are represented by the parameters defined in Section 2.4.2.1. In the static threshold's case, let c denote the constant threshold, then

False-positive: $\Pr(R_0 \geq c) = 1 - F_{conn0}(\lceil c-1 \rceil)$

False-negative: $\Pr(R_1 < c) = F_{conn1}(\lceil c-1 \rceil)$

False-positive and false-negative equations for per-process containment are calculated as follows. The threshold in this case is a random variable, rather than a constant, since L is a random variable and $h(L) = T$.

False-positive:

$$\Pr(R_0 \geq T_0) = \Pr(T_0 \leq R_0) = \sum_{r=0}^{\infty} \Pr(T_0 \leq r) \Pr(R_0 = r)$$
$$= \sum_{r=0}^{\infty} F_{T0}(r) f_{conn0}(r). \tag{2.1}$$

Note that F_{T0} is the CDF of T_0 when the process is normal and $h(l) = t$. We also know L_0 has CDF F_0 and pdf f_0 for normal processes. By the Change of Variables Theorem [4],

$$f_{T0}(t) = f_0(h^{-1}(t))\frac{1}{h'(h^{-1}(t))} \; t \in [h(1), h(0)]$$

$$F_{T0}(t) = \begin{cases} 0 & t < h(1) \\ \int_{h(1)}^{t} f_0(h^{-1}(t))\frac{1}{h'(h^{-1}(t))}\mathrm{d}t & t \in [h(1), h(0)] \\ l & t > h(0) \end{cases}$$

Similarly, false-negative:

$$\begin{aligned} \Pr(R_1 < T_1) = \Pr(T_1 > R_1) &= \sum_{r=0}^{\infty} \Pr(T_1 > r)\Pr(R_1 = r) \\ &= \sum_{r=0}^{\infty}(1 - F_{T1}(r))f_{conn1}(r). \end{aligned} \tag{2.2}$$

$$F_{T1}(t) = \begin{cases} 0 & t < h(1) \\ \int_{h(1)}^{t} f_1(h^{-1}(t))\frac{1}{h'(h^{-1}(t))}\mathrm{d}t & t \in [h(1), h(0)] \\ l & t > h(0) \end{cases}$$

Each pair of a and c determines a mapping function. We plug in the connection-rate and suspicion-level distributions as well as the mapping function into the equations (2.1) and (2.2) to calculate the false-positive and false-negative rates. Since a pair of a and c corresponds to a pair of false-positive and false-negative, by varying the values of a and c, we can plot a set of false-alarm profiles. As shown in Figure 2.4, given a, as the value of c decreases from 14 to 1, the curve descends from the upper-left to lower-right direction meaning that the more restrictive the threshold (the smaller the c), the higher the false-positive and the lower the false-negative, showing a tradeoff between the two. Fixing the same set of c from 1 to 14, we vary

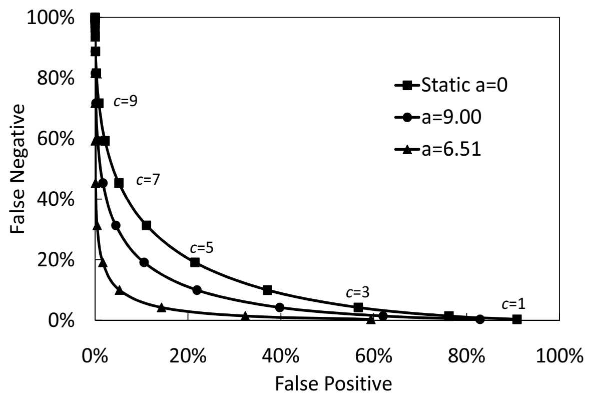


Figure 2.4: Static threshold vs. per-process false alarm profiles

a's value and generate a curve for each a. When $a > 0$ (we only draw $a = 6.51$ and $a = 9$ for illustration), meaning that we make use of the suspicion-level information and assign each process a customized threshold, the curves are always below the static threshold approach (i.e., $a = 0$). In other words, given a false-positive rate, the $a > 0$ curves can always achieve lower false-negative rates than $a = 0$ curve does, indicating that using per-process suspicion-level information results in an improved false alarm curve.

2.4.3 Mapping Function Optimization

To find the most appropriate mapping function for a specific host system, we develop an optimization algorithm. The required false-positive rate is the input to the optimization algorithm that determines a and c to obtain the lowest false-negative rate. We impose a constraint on the false-positive rate because users are affected most by this rate. But this configuration is tunable such that false-negative rate could also be constrained. Since it is difficult to derive explicit equations for a and c, we devise a heuristic algorithm based on the observation of the numerically-obtained false alarm curves. One property of the curve is that given a, the larger the c, the lower the false-positive rate. Another property is that given c, there is an optimal a that could achieve the lowest false-positive. Our algorithm consists of adjustment (steps 1–3) and refinement (steps 4–6). The first phase searches for the curve to the lowest-left direction, while the second phase helps to jump out of the local optimum facing the first phase, if any.

1. (Adjust): given the initial a and c, increase or decrease c to achieve the target false-positive rate.

2. (Adjust): fix c at the value obtained from step 1, increase or decrease a to reach the lowest false-positive rate.

3. (Adjust): repeat steps 1 and 2 until a cannot be changed any further.

4. (Refine): increase or decrease a if a lower false-negative rate can be achieved.

5. (Refine): adjust c to the target false-positive rate.

6. (Refine): repeat steps 4 and 5 until the lowest false-negative rate is reached.

The pseudocode for this algorithm is given below.

```
ADJUST_C($\&a, \&c, F_{conn0}, targetFP$)
1  if FP($a, c, F_{conn0}$) $> targetFP$
2     then $step \leftarrow EPSILON$
3     else $step \leftarrow -EPSILON$
4  while FP($a, c, F_{conn0}$) $> targetFP$
5       do $c \leftarrow c + step$ or $c \leftarrow c - step$
6  ADJUST_A($a, c, F_{conn0}, targetFP$)
```

```
ADJUST_A($\&a, \&c, F_{conn0}, targetFP$)
1  $currenta \leftarrow a$
2  $currentFP \leftarrow$ FP($a, c, F_{conn0}$)
3  if FP($a + step, c, F_{conn0}$) $< currentFP$
4     then $step \leftarrow EPSILON$
```

```
 5    else  $step \leftarrow -EPSILON$
 6  while FP$(a + step, c, F_{conn0}) < currentFP$
 7      do $a \leftarrow a + step$
 8         $currentFP \leftarrow temp$
 9  if $a - currenta < EPSILON$
10    then $return$
11  ADJUST_C$(a, c, F_{conn0}, targetFP)$
```

```
REFINE_A$(\&a, \&c, F_{conn0}, F_{conn1}, targetFP)$
 1  repeat
 2          $currentFN \leftarrow$ FN$(a, c, F_{conn1})$
 3          if FN$(a + EPSILON, c, F_{conn1}) < currentFN$
 4            then $a \leftarrow a + EPSILON$
 5          if FN$(a - EPSILON, c, F_{conn1}) < currentFN$
 6            then $a \leftarrow a - EPSILON$
 7          if FN$(a + EPSILON, c, F_{conn1}) == currentFN$
 8            then $return$
 9          REFINE_C$(a, c, F_{conn0}, targetFP)$
10    until FN$(a, c, F_{conn1}) > currentFN$
```

```
REFINE_C$(\&a, \&c, F_{conn0}, targetFP)$
 1  if FP$(a, c, F_{conn0}) > targetFP$
 2    then $step \leftarrow EPSILON$
 3    else  $step \leftarrow -EPSILON$
 4  while FP$(a, c, F_{conn0}) > targetFP$
 5      do $c \leftarrow c + step$ or $c \leftarrow c - step$
```

The EPSILON is set to be 10^{-2} and the call sequence is:

adjust_c(a,c,F_{conn0},targetFP)

refine_a(a,c,F_{conn0}, F_{conn1},targetFP)

2.5 Implementation and Evaluation

We have implemented four behavior monitors: Registry Activity Monitor (RAM), File Activity Monitor (FAM), Network Activity Monitor (NAM), and Process Correlation Monitor (PCM). These monitors capture each process's behavior in real time. The traces collected by these monitors are fed to our trace-driven simulation of the proposed framework. The traces were collected from 20 real-world worms plus some of their variants that are representative and reflect the evolution of contemporary worms and 49 normal programs. We used a C++ implementation of SVM learning algorithm, called LibSVM [31], in the behavior analysis component and derived suspicion-level distributions for normal and malicious processes. We also tested the containment scheme's false-positive and false-negative rates in evaluation.

2.5.1 System Monitors

The architecture of RAM resembles that of Sysinternals' Regmon [18]. RAM was implemented on Windows NT/XP, including a user-level logging application and a kernel device driver which implemented the system-call hooking technique [2]. RAM intercepts registry-related system calls and stores passed parameters and other status information in a kernel buffer which is then periodically copied to the user-level application. RAM logs complete information about every registry activity for all processes running on a host, including timestamp, process name, process ID, request type (create key, set key value, etc.) and path of the registry key. The implementation of FAM is similar to that of RAM. It records system-wide file-system activities in real time. NAM is implemented based on WinPcap library, and continually monitors all incoming and outgoing packets of the host. With WinPcap's support, NAM provides information on active connections (e.g., source address, port, destination address port, process ID, etc.) and dynamically correlates each captured packet with the process that initiates this connection. The data collected by NAM consists of timestamp, process name and ID, connection type (TCP or UDP) and detailed packet header. PCM uses the same technique as Sysinternals' Process Explorer [16]. The idea is to call a Windows Native API named *NtQuerySystemInformation*. This API retrieves an array of SYSTEM_PROCESS_INFORMATION structures for processing running in the system, in which each process's parent ID can be obtained. Another set of Windows APIs, Process Structure Routines, are used to track process creations and terminations. These four monitors together characterize the detailed behavior of all the running processes, which will be formalized into behavior feature vectors to determine the per-process suspicion level by the machine learning algorithm.

2.5.2 Trace Collection

To collect worm traces in real time, we set up 3 virtual machines running Windows XP systems connected via a virtual network as our test-bed. We also set up a DNS server at the host machine to collect DNS statistics and configured it as the default gateway for the virtual machines. By studying recent worm behaviors, we selected 20 real-world worms and their variants. The samples include notable worms such as Blaster, MyDoom, Storm, etc. We ran the worm samples on our test-bed, gathering their process correlation, file system, registry and network activities. The length of trace for each worm is approximately 20 minutes. The normal traces were collected from malware-free PCs in regular use. We selected applications with network access, such as P2P, web browser, file download, etc. The traces captured the activities of 49 normal processes which cover most commonly-used network applications, including eMule, IE, firefox, sshclient, utorrent, etc., and each lasted 20 minutes as well. We did not capture longer traces because most applications show relatively stable behavior. We used part of both normal and worm traces to train the SVM to build profiles and the rest as our test set. We intentionally selected variants of some worm samples into the test set and the original worms in the training set. The accuracy of the learning algorithm with regard to suspicion-level generation is demonstrated in Section 2.5.4.

2.5.3 Trace Formalization

We extracted useful features from the file system, registry and network activity logs, and formalized them to feature vectors in a uniform format that can be analyzed by the learning algorithm. A feature vector has 15 dimensions, each of which corresponds to an atomic behavior feature represented with a tuple <feature index:value>. A detailed description of the behavior feature vector is given in Section 2.3.1. As described earlier, we kept track of the process relationships. The behavior features are aggregated across correlated processes. For example, process A registers an autorun entry in the registry and creates process B. Process B then drops a backdoor in the system directory within the same time window. Suppose the behavior vector's first two fields are $\langle autorun\ key, dll\ drop, \ldots\rangle$. Then, the vector in this window for both will be $\langle 1, 1, \ldots\rangle$ instead of $\langle 1, 0, \ldots\rangle$ and $\langle 0, 1, \ldots\rangle$.

In addition, we selectively accumulated some feature fields in consideration of the worms that may spread malicious activities to different time windows. Because we did not find such a worm in the wild, we simulated one to show the difference in suspicion levels. Without feature accumulation, the suspicion level ranges from 0.027 to 0.59, and 7 out of 8 are below 0.5 (Table 2.3). With feature accumulation over 5 time windows, the suspicion level becomes 0.89. Note that the simulated worm is slowly propagated compared to fast-scanning worms that can generate hundreds

Table 2.3: Suspicion levels with and without feature accumulation

Time Window	Behavior	Original SusLevel	Accumulated SusLevel
win0	Write exe to sys dir	0.59	0.59
win1	Write a dll	0.36	0.68
win2	Create an autorun key	0.20	0.72
win3	Create an Inprocserver key	0.22	0.80
win4	Initiate 20 connections	0.16	0.89
win5	None	0.027	0.89
win6	Initiate 20 connections	0.16	0.89
win7	None	0.027	0.89

of connections in a time window. Even so, due to its accumulated file system and Registry activities, its suspicion level is high enough to trigger a strict rate-limiting when it accesses the network.

For this feature accumulation, we need to determine how long to accumulate behavior for each process. If this time window of accumulation is static, an attacker may learn and evade it. If we accumulate over infinitely many windows, a total number of false-positives may become very high. So, we dynamically change the accumulation time window. For example, we randomly select a value between 1 and 50 min each time. By introducing this uncertainty, it makes evasion of behavior monitoring harder.

2.5.4 Suspicion-Level Analysis

Recall that the suspicion level generated by the learning algorithm is denoted by L_0 for normal and L_1 for malicious processes where $L_0, L_1 \in [0, 1]$. L_0 (L_1) has CDF F_0 (F_1). We estimated the suspicion-level CDFs for normal and malicious processes by applying the pre-trained SVM to the behavior vectors generated from the normal and malicious traces. The suspicion-level CDFs are plotted in Figure 2.5. Clearly, normal applications tend to have a lower degree of suspicion, while worms have much higher suspicion levels. This demonstrates the SVM's learning ability in determining the suspicion level of a process, and also indicates that the thus-generated suspicion

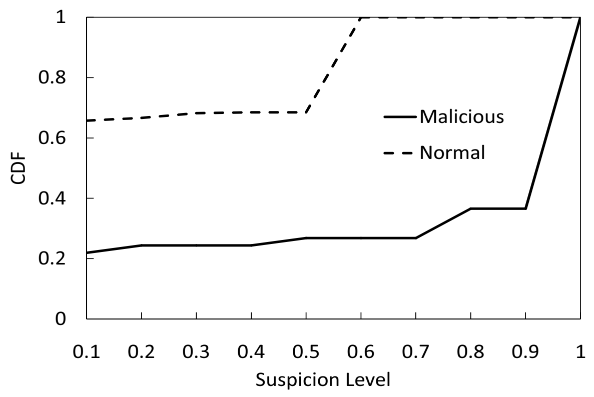


Figure 2.5: Suspicion-level CDFs for malicious/normal processes

level is indeed informative.

2.5.5 Overhead

One may want to know the overheads incurred by the runtime behavior capture and the periodical suspicion-level generation. We used a common Windows benchmark PassMark Software, PerformanceTest [14], to measure the overheads of the runtime system monitors on a host machine with Intel(R) Pentium IV 1.5GHz CPU, 512MB memory, 19.5G disk, and Windows XP operating system. We ran the corresponding benchmark program for CPU, memory and disk, respectively, 5 rounds each. The average overhead for CPU is 10.5%, memory 14.5% and disk 4.7%. Considering the fact that memory of this machine is much smaller than that of a today's PC/laptop, all of these numbers are within an acceptable range. As to the suspicion-level generation, since the classifier is pre-trained (i.e., the support vectors are pre-loaded), the training time will not incur any runtime overhead to the host. Calculation of the suspicion level is fast. For example, the suspicion levels of 10 processes are generated within half a second.

2.5.6 Trace-Driven Evaluation

We simulated the running of different worms and normal processes based on the real traces collected. To demonstrate the efficacy of our scheme, Williamson's rate-limiting [93] was implemented as a baseline in our evaluation. We specifically compared the performance between Williamson's and our per-process schemes when a host was infected by the latest Storm worm. We also applied Williamson's, static-threshold (i.e., processes have the same threshold) and our customized per-process to all other test-set data we collected including worms, their variants and normal programs, to show the performance differences.

2.5.6.1 Case Study: Storm Worm

Storm worm (or W32.Peacomm, Nuwar, Zhelatin) spreads via email spam and is known to be the first malware to seed a botnet in a P2P manner without any centralized control. The instance we obtained was from the Storm outbreak on Valentine's Day 2008. The trace shows that Storm first connects to the P2P network by contacting peers in a hard-coded peer list containing more than 100 IPs. After joining the network, the bot sends out search requests to find a specific secondary injection for spamming. We observed that it started to behave as a SMTP server and to send spam email in 5 minutes upon execution.

In our experiment, we applied both Williamson's rate-limiting and our per-process schemes. Williamson's approach applied to the entire host. A working set of specified size ($n = 4$ in our case, as commonly used by others) is maintained to keep track of all IPs the host has contacted. When a new connection is initiated, the destination IP is compared with those in the working set. If it is in that set, the connection can pass through. Otherwise, it is placed in a delay queue and will be sent out later. At periodic intervals (every second as Williamson proposed), one connection is dequeued and a new destination address is added to the working set. There is a pre-determined

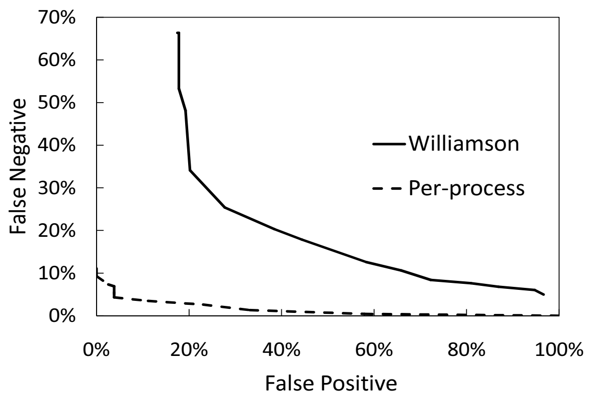


Figure 2.6: False alarm profiles for Storm worm under Williamson and our approaches

threshold for the delay queue. Whenever this value is exceeded, all new connections are dropped.

We mixed the normal trace including P2P, web browser applications with storm trace in the simulation of Williamson's per-host scheme. We varied the delay queue threshold in each round to get a pair of false-positive and false-negative rates. In our per-process scheme, we used the mapping function $h(l) = c + 0.5 * a - a * l$. The optimal mapping is generated by the algorithm described in Section 2.4. We obtained several pairs of false-positives and false-negatives via different mapping functions and then drew the false alarm profiles. Figure 2.6 compares false alarms of the two approaches, showing that our scheme outperforms Williamson's. In Williamson's scheme, when the delay queue threshold is set to a small value, Storm can saturate the delay queue in tens of seconds, thus causing normal traffic to be dropped. When there are some network-intensive normal applications, the false-positive is considerable. Specifically in the experiment, the false-negative rate is controlled within 10% at the expense of more than 70% false-positive rate. On the other hand, given a generous delay queue threshold to accommodate normal traffic, a majority of Storm's connections can pass through too. The problem lies in treating all processes indiscriminately. In per-process scheme, different thresholds are assigned to different processes according to their suspicion levels. In the normal case, the average suspicion level for all normal applications is around 0.3 despite that some network-intensive applications are included. On the other hand, Storm has acted maliciously at the Registry and file system resulting in a suspicion level of 0.95 and hence a low rate-limiting threshold. By imposing customized rate-limiting to Storm and normal traffic, our scheme achieves lower false-positive and false-negative rates.

2.5.6.2 Evaluation of Three Schemes

Note that Williamson's use of a per-host static threshold is slightly different from the static-threshold approach mentioned before. The latter assigns the same threshold to all processes, while the former does not discriminate at the process level. To compare these two and our schemes, either false-positive or false-negative rate has to be fixed as there is always a tradeoff between the two. We set the false-positive rate to be 5% and input the parameters to the optimization algorithm. The resulting optimal mapping function, which can generate the lowest false-negative rate is $h(l) = 7.32 + 0.5 * 6.51 - 6.51 * l(a = 6.51, c = 7.32)$. Based on the numerically-obtained false alarm profiles introduced in Figure 2.4, the static threshold rate is set to 8 per second to meet the false-positive rate requirement. Since a suspicion level is generated for each process every t minutes (t is set to 1 in our experiment), the threshold value calculated by the mapping function is also updated dynamically in the order of minutes. While applying Williamson's scheme, we chose the well-adopted parameters (working set length=4 and release rate of delay queue= 1 connection/s). We varied the delay queue threshold imposed to mingled normal and worm traces to find the one that can reach a 5% false-positive rate. The appropriate value is found to be 200.

Recall that the false-positive rate is defined as the fraction of normal connections blocked. Figure 2.7 plots real false-positive rates for each normal program in the test set under Williamson's, the static threshold, and the customized per-process schemes. They have some fluctuations process-wise but the average values are close

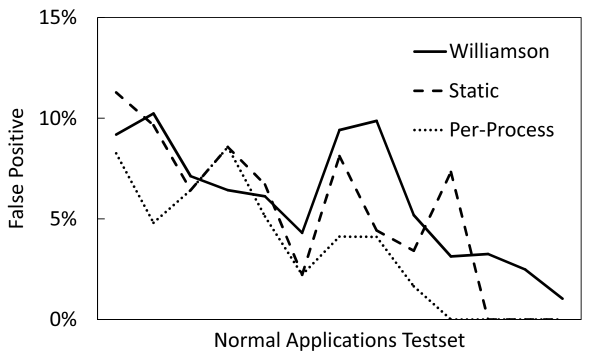


Figure 2.7: False-positives on normal applications

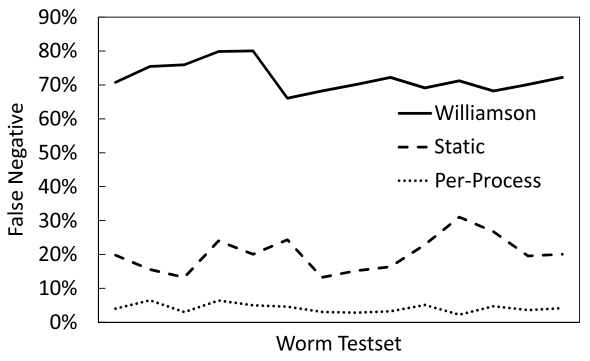


Figure 2.8: False-negatives on worms

to 5% (see Table 2.4). Figure 2.8 shows the false-negative curve for each scheme. The false-negative rate is calculated as the percentage of evaded connections of worms. Given false-positive rates shown in Table 2.4, the average false-negative rate across all worms under Williamson's scheme is the highest, 72.12%, and the per-process scheme 4.15%, the best. The static scheme produces 20.14%, in the middle. Williamson's is even worse than the static threshold scheme for the following reason. During a worm outbreak, the per-host delay queue is mostly occupied by the worm. When the threshold is set to a larger value to accommodate normal applications, the worm benefits more, leading to a high false-negative rate. As for the static threshold scheme, the threshold is assigned to each process, which is relatively small (in our case, 8 versus 200) and thus more restrictive, compared to the per-host threshold.

Although progress is made from the per-host to the static threshold scheme, our customized per-process threshold can perform even better than the static threshold

Table 2.4: Average FPs and FNs under three schemes

	Avg FP	Avg FN
Williamson	5.56%	72.12%
Static	4.87%	20.14%
Per-Process	3.24%	4.15%

scheme by using suspicion-level information. The suspicion levels for worms are relatively high and their thresholds are accordingly low so that most of malicious connections may be blocked, whereas normal applications are assigned low suspicion levels but high rate-limiting thresholds to let most traffic pass through. Compared to the customized threshold, the static threshold scheme must compromise one false alarm rate for another. Thus, the customized per-process scheme performs best among the three.

2.5.6.3 Optimality of the Mapping Function

We now want to show that the performance improvement from use of a static threshold to a customized per-process threshold is not a coincidence and that the mapping function used is optimal in the sense that it can achieve the lowest false-negative rate given a false positive rate. We selected 12 pairs of a and c, i.e., 12 mapping functions, and measured the false-positives and false-negatives on the same data set. Figure 2.9 plots the false-alarm profiles for different a and c values based on the real-world traces. As we expected, the curves are quite similar to the numerically-computed false-alarm profiles (Figure 2.4). The curve in the lower-left direction is the one with the optimal a because given a false-positive rate, the false-negative rate on this curve is always lower than other curves, and it is the same case when false-negative is fixed. In this figure, $a = 6.51$ is obviously better than smaller or larger values of a. This is the value our optimization algorithm yielded. In particular, on this curve, $c = 7.32$ is the one close to our required false-positive rate 5%. This c value is also identical to that generated from the optimization. Moreover, when $a = 0$ which represents the static scheme, the false-negatives on this curve are generally higher than those on

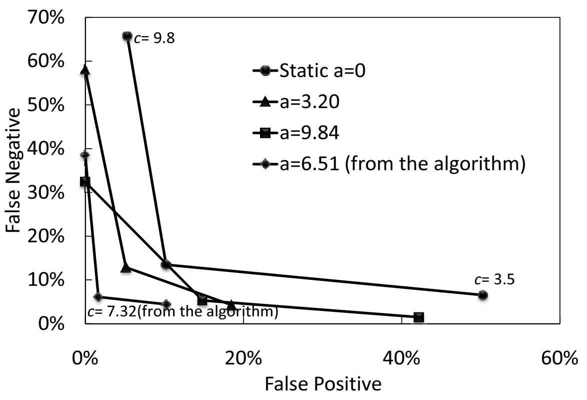


Figure 2.9: Optimality of the mapping function

other curves. All of these observations confirm that our empirical results obtained from real-world traces are consistent with the numerical results obtained from false-alarm modeling, indicating that the mapping function selected by the optimization algorithm is indeed optimal and the per-process scheme performs significantly better than the static threshold scheme in terms of false-alarm rates.

2.6 Limitations and Fixes

In this section, we would like to discuss two fundamental limitations of not only our scheme but all host- and behavior-based worm defense and response mechanisms. The first limitation is the circumvention of a pre-defined list of behaviors. Since our behavior list that can best discriminate normal and malicious programs is based on the study of existing network worms, our scheme works effectively for malicious processes having typical "worm" behavior. Even if some worms change their behaviors a little bit, such as installing in a different directory, our scheme can still work since we account for a *set* of behavior features, not just one feature. As worms evolve, we can simply extend or modify our behavior list of monitoring. However, if all of the behaviors of a worm are the same as those of normal programs or completely different from existing worms, we can hardly capture it. However, such a scenario will be rare as our behavior list reflects the fundamentals of network worm behavior.

The second limitation is the vulnerability of a host-based mechanism to worms' adaptivity. As mentioned before, there is always a tradeoff between deploying worm defense at the network and at the end-host systems. A network-based scheme is not easy to be disabled by a worm but only gets coarser-grained information, i.e., network activities on a host basis, resulting in less accurate and efficient response. A host-based solution, on the other hand, can obtain finer-grained information and achieve finer-grained and accurate response as we have demonstrated. This guarantees that critical services and applications will remain uninterrupted even during an worm outbreak. Moreover, since in-host monitors keep track of run-time activities for each process, when and how a system is infected can be traced back for further investigation. However, a worm can get around our scheme by sitting below the monitoring level and modifying or subverting the information our monitor receives by using the rootkit technique for example. One countermeasure is to search for the discrepancy between the information returned by the Windows API or system calls and that seen in the raw scan of the file system or Registry hive [20]. With the help of secure hardware [26] or secure VMM [41], it is also possible to prevent or detect the rootkit from altering the OS. Another possible evasion is that a worm can employ benign processes to conduct malicious activities. For example, upon execution, a worm drops some DLL files into the file system and registers a certain Registry key so that benign programs will automatically load these DLLs and do malicious activities for the worm. In this case, simply tracking process creations and parent-child relationships will not reveal the correlation between the benign process and the worm process. One possible solution is to link the process that registers such a Registry entry with the process that uses this entry, considering that our scheme already tracks the process injection behavior at the Registry. This would add additional complexity to our framework. We plan to study its feasibility and implications in the future.

While there is a possibility that a worm could attempt to evade our mechanism, in the evaluation section we have demonstrated how our system successfully contains state-of-the-art worms that we were able to collet while minimizing the impact on legitimate traffic. Therefore, our approach at least raises the bar significantly for contemporary network worms.

2.7 Conclusion

We have proposed a novel automatic worm defense framework that combines per-process behavior analysis and fine-grained containment. It automatically monitors each process's runtime behavior and generates its level of suspicion by a machine learning algorithm. A mapping algorithm is developed to transform the suspicion level to the appropriate rate-limiting threshold on a per-process basis. Our experimental evaluation based on real-world worm samples and normal process traces demonstrates the efficacy of per-process rate-limiting, which produces much fewer false-positives and false-negatives in containing network worms than previously-known approaches. In the future work, we would like to further investigate the evasion schemes an advanced worm can utilize to get around our approach and how to defeat such schemes. For example, it is worth examining how to extend our framework to identify benign processes that are exploited by worms to conduct malicious tasks and correlate them with those worm processes.

CHAPTER III

Botnet Detection Using Combined Host- and Network-Level Information in the Edge Network

3.1 Introduction

Botnets have now become one of the most serious security threats to Internet services and applications. Botnets can cooperatively mount Distributed-Denial-of-Service (DDoS) attacks, spamming, phishing, identity theft, and other cyber crimes. To control a botnet, a botmaster needs a C&C channel to issue commands, and coordinate bots' actions. Traditional botnets utilize the IRC or HTTP protocol as their C&C infrastructure, which are vulnerable to a single-point-of-failure. That is, once the central IRC or HTTP servers are identified and removed, the entire botnet will be disabled. To overcome this weakness, attackers have recently shifted toward a new generation of botnets utilizing decentralized C&C protocols such as P2P. This C&C infrastructure makes detection and mitigation much harder. A well-known example is the Storm worm [22] which spread via email spam and is known to be the first malware to seed a botnet in a P2P fashion. The first Storm worm came out in early 2007 and the attackers then constantly changed the malicious code to create multiple variants. Another spambot Waledac, which came to the wild at the end of 2008, also spread via spam emails and formed its botnet using a C&C structure similar to that of the Storm botnet. Some researchers pointed out that Waledac was the new and improved version of the Storm botnet [67].

To date, most botnet-detection approaches operate at the network level; a majority of them target traditional IRC- or HTTP-based botnets [29, 43, 45, 56, 60, 83] by looking for traffic signatures or flow patterns. We are aware of only one approach [44] designed for protocol- and structure-independent botnet detection. This approach requires packet-level inspection and depends solely on network traffic analysis, unlikely to have a complete view of botnets' behavior. We thus need the finer-grained host-by-host behavior inspection to complement the network analysis. On the other hand, since bots behave maliciously system-wide, general host-based detection can be useful. One such way is to match malware signatures, but it is effective in detecting known bots only. To deal with unknown bot infiltration, in-host behavior analysis [34, 39, 57, 75, 80] is needed. However, since some in-host malicious behavior is not exclusive to bots and in-host mechanisms are vulnerable to host-resident malware, host-based approaches alone can hardly provide reliable detection results and thus we need external, hard-to-compromise (i.e., network-level) information for more accurate detection of bots' malicious behavior.

Considering the required coordination within each botnet at the network level and the malicious behavior each bot exhibits at the host level, we propose a C&C protocol-independent detection framework that incorporates information collected at both the host and the network levels. The two sources of information complement each other in making detection decisions. This framework is intended for use in edge networks such as enterprise networks as it requires fine-grained information for analysis. The framework first identifies suspicious hosts by discovering similar behaviors among different hosts using network- flow analysis, and validates the identified suspects to be malicious by scrutinizing their in-host behavior. Since bots within the same botnet are likely to receive the same input from the botmaster and take similar actions, whereas benign hosts rarely demonstrate such correlated behavior, our framework looks for flows with similar patterns and labels them as triggering flows. It then associates all subsequent flows (action flows) with each triggering flow on a host-by-host basis, checking the similarity among those associated groups. If multiple hosts behave similarly in the trigger-action patterns, they are grouped into the same suspicious cluster as likely to belong to the same botnet. Whenever a group of hosts are identified as suspicious by the network analysis, the host-behavior analysis results based on a history of monitored host behaviors are reported. A correlation algorithm finally assigns a detection score to each host under inspection by considering both network and host behaviors.

3.1.1 Contributions

Our work makes the following contributions. First, to the best of our knowledge, this is the first framework that combines both network- and host-level information to detect botnets. The benefit is that it completes a detection picture by considering not only the coordination behavior intrinsic to each botnet but also each bot's in-host behavior. Moreover, we extract features from NetFlow data to analyze the similarity or dissimilarity of network behavior without inspecting each packet's payload, preserving privacy. Second, our detection relies on the invariant properties of botnets' network and host behaviors, which are independent of the underlying C&C protocol. It can detect both traditional IRC and HTTP, as well as recent P2P botnets. Third, our approach was evaluated by using several days of real-world NetFlow data from a core router of a major campus network containing benign and botnet traces, as well as multiple benign and botnet data sets collected from virtual machines and regular hosts. Our evaluation results show that the proposed framework can detect different types of botnets with low false-alarm rates.

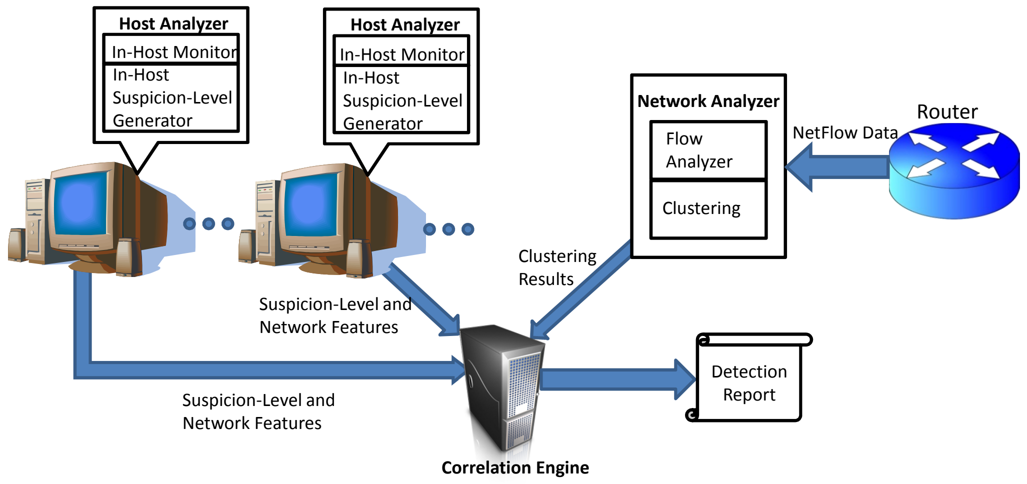


Figure 3.1: System architecture

3.1.2 Organization

This chapter is organized as follows. Section 3.2 provides an overview of our system architecture. Section 4.3 presents the proposed detection methodology and implementation details. Section 4.4 demonstrates evaluation results. Section 3.5 discusses limitations. The chapter concludes with Section 5.6.

3.2 System Architecture

Figure 4.1 shows the architecture of our system, which primarily consists of three components: host analyzer, network analyzer, and correlation engine.

As almost all of current botnets target Windows machines, our host analyzer is designed and implemented for Windows platforms. The host analyzer is deployed at each host and contains two modules: in-host monitor and suspicion-level generator. The former monitors run-time system-wide behavior taking place in the Registry, file system, and network stack on a host. The latter generates a suspicion-level by applying a machine-learning algorithm based on the behavior reported at each time window and computes the overall suspicion-level using a moving average algorithm. The host analyzer sends the average suspicion-level along with a few network feature statistics to the correlation engine, if required. The network analyzer also contains two modules: flow analyzer and clustering. The flow analyzer takes the flow data from a router as input and searches for trigger-action botnet-like flow patterns among different hosts. It then extracts a set of features that can best represent those associated flows and transforms them into feature vectors. Those vectors are then fed to the clustering module that groups similarly-behaving hosts into the same cluster, assuming them likely to be part of a botnet. Whenever a suspicious group of hosts are identified by the network analyzer, their host analyzers are required to provide the suspicion-level and network statistics to the correlation engine, which verifies the validity of the host information by comparing the network statistics collected from the network and those received from the host. The correlation engine finally assigns a detection score to each host and produces a detection result.

3.3 Methodology and Implementation

Our framework consists of three main components: host analyzer, network analyzer, and correlation engine. Each of these components is detailed next.

3.3.1 Host Analyzer

The host-analyzer is composed of two modules: in-host monitor and in-host suspicion-level generator.

3.3.1.1 In-Host Monitor

Each in-host monitor captures system-wide behavior in real time at different locations. By studying contemporary bots' behaviors, we have observed that they share certain behavior patterns that are different from benign applications, and that their behaviors can be grouped into 3 categories taking place at the Registry, file system and network stack. For example, when infecting a computer, a bot first creates an exe or dll file in the system directory. It then registers an autorun key in the Registry to make itself run automatically whenever the host system boots up. It also injects its code into other processes to hide its presence and disables anti-virus software and the task manager, if necessary. Finally, it opens one or more ports for further communications and establishes connections with the botmaster or peers in order to launch DDoS, spamming activities, etc. Note that a single activity mentioned above may not be malicious because it is also likely to be performed by benign hosts. However, the combination and aggregation of these activities can reveal that a host has been infected, since chances are slim that a benign host conducts all of these activities. Thus, the in-host suspicion-level analysis considers the behavior features altogether while making decisions. The implementation of the in-host monitors was adapted from the per-process monitors used in our previous work [97]. Every in-host monitor consists of three sub-monitors. The sub-monitors at the Registry and file system implemented system-call hooking that intercepts related-system calls, stores the passed parameters and status information in a kernel buffer, and then copies them to the user-level application. The two sub-monitors log complete information of every activity of interest, including timestamp, request type and path. The sub-monitor at the network stack was implemented based on WinPcap library and monitors all incoming and outgoing traffic of the host. It collects information including source and destination IPs, ports, and the protocol.

To facilitate a further analysis, each host's run-time behavior is transformed into a uniform format known as a *behavior vector*. Each behavior vector consists of 9 behavior features as shown in Table 3.1; these features are intrinsic to bot-infected hosts. Each feature is represented by a tuple <feature index:value>. For example, the first tuple below means the host created 2 files in the system directory.
1:2 2:2 3:1 4:1 5:2 6:3 7:40 8:55 9:40 [00:10:51, 01:10:51] As each host's network activities can be captured and analyzed at the network level, the in-host monitor should focus on behaviors that can't be observed externally, such as file and Registry

Table 3.1: In-host behavior features

Index	Behavior Features
1	DLL or EXE Creation in System Directory
2	Modification of Files in System Directory
3	Creation of AutoRun Key in Registry
4	Creation of Process Injection Key in Registry
5	Modification of Critical Registry Key (Disabling taskmgr; Overriding antivirus, etc.)
6	Number of Ports Opened
7	Number of Suspicious Ports
8	Number of Unique IPs Contacted
9	Number of SMTP Flows

operations, to complement the network-level information. However, since a host is vulnerable to being compromised, we need some information that can be obtained both internally and externally to validate the integrity of the data provided by a host. Therefore, we have added a few network features (feature 7 to 9) for in-host monitoring; these features will be compared against the same features generated by the network-level analyzer.

3.3.1.2 In-Host Suspicion-Level Generator

Given each host's behavior vector, we employ a supervised learning algorithm, or the *support vector machine* (SVM), to quantify its suspicion level. SVM learns from benign and malicious host behavior profiles prior to predicting unlabeled behavior vectors. Since bots' in-host behaviors are similar to other types of malware such as network worms, we did not confine our training data to bot-infected hosts but also included other malware-infected ones. Benign hosts' training traces were obtained directly from malware-free hosts in normal use. Based on the training data, the SVM creates a hyperplane corresponding to a classification rule. Given a new behavior vector, the SVM estimates the distance of the sample from the hyperplane and decides which class it belongs to. Note that the training data were completely different from the test set in the evaluation. To make the most of this learning model, we calibrate the distance score to a posterior classification probability indicating how likely a test behavior vector belongs to a particular class [59]. The posterior probability is then translated into the suspicion level in [0, 1] where 0 is benign and 1 is bot-infected. The higher the suspicion level, the more likely it is bot-infected.

Since the suspicion level for each host is generated at every time window, a bot may intentionally reduce its suspicion level by spreading malicious activities into different time windows or even sleeping for a while. To counter such an evasion attempt, we selectively accumulate the value in each field of the behavior feature vector. The features worth accumulation are those typical to bot-infected hosts, such as creating an autorun key in the Registry or injecting a piece of code into another process. In addition, we use the Exponential Weighted Moving Average (EWMA) algorithm to compute the average suspicion level at every time window. If Y_n denotes the suspicion level generated in the n-th time window, and S_{n-1} is the estimated average suspicion level at the $(n-1)$-th window, the estimated average at the n-th window is given by $S_n = \alpha * Y_n + (1-\alpha) * S_{n-1}$ where α is a constant smoothing factor. We define α as a function of the time interval between two suspicion-level readings. $\alpha = 1 - e^{-\frac{t_n - t_{n-1}}{W}}$ where $t_n - t_{n-1}$ is the length of the time window of generating suspicion levels and W is the the period of time over which the suspicion level is averaged. We chose $t_n - t_{n-1} = 10$ and $W = 60$ minutes, meaning that the in-host generator produces a suspicion level every 10 minutes and reports the average to the correlation engine on an hourly basis. The moving average is thus expressed as

$$S_n = (1 - e^{-\frac{1}{6}}) * Y_n + e^{-\frac{1}{6}} * S_{n-1}. \tag{3.1}$$

3.3.2 Network Analyzer

Considering privacy concerns and computational costs, our network analyzer, which operates on the network traffic collected from a core router in a major cam-

Table 3.2: Flow features

Index	Flow Features
1 to 4	Duration Mean, Variance, Skewness and Kurtosis
5 to 8	Totalbytes Mean, Variance, Skewness and Kurtosis
9 to 12	Number of Packets Mean, Variance, Skewness and Kurtosis
13	Number of TCP Flows
14	Number of UDP Flows
15	Number of SMTP Flows
16	Number of Unique IPs Contacted
17	Number of Suspicious Ports

pus network, only requires analysis of NetFlow [13] data without accessing packets' payload. NetFlow is a network protocol developed by Cisco for summarizing IP traffic information . A flow is defined as a sequence of packets between a source and a destination within a single session or connection. A NetFlow record contains a variety of flow-level information, such as protocol, source/destination IP and port, start and end timestamps, number of packets, and flow size, but has no packet content information. The network analyzer takes flow records from the router as input and generates host-clustering results. It consists of two modules: flow analyzer and clustering, which were implemented in Perl and R.

3.3.2.1 Flow Analyzer

The flow analyzer processes the flow records of all hosts in a network to extract trigger-action patterns of interest. Recall that bots within the same botnet usually receive the same input from botmasters and take similar actions thereafter. Such coordinated behaviors are essential and invariant to all types of botnets regardless of their C&C structures.

The first step in flow processing is to filter out irrelevant flows including internal flows and legitimate flows. Internal flows represent traffic within a network. Legitimate flows are those with well-known destination addresses such as Google and CNN which seldom function as C&C servers. Note that flow filtering is just an optional operation and not essential to our network analyzer. It is only used to reduce the total number of flow records, and thus, the computational cost. It turns out that the NetFlow data obtained from the core router in our campus network contained an average of 3,900,000 flows per hour, and that the number could be reduced to an average of 25,000 flows including 2,000 hosts per hour after filtering.

In the second step, our analyzer searches for trigger-action patterns at each time window. In the monitored network, it looks for suspicious flows with the same destination IP and protocol across all hosts which are presumably receiving commands, and labels them as triggering flows. We found that bots within the same botnet all connect to the same set of IPs. Evidently, IRC- and HTTP-based bots talk to their C&C servers. In the hybrid-P2P-based case, Storm instances bootstrap by connecting to the IPs in a hard-coded list, making their contacted IP lists look alike. Waledac instances demonstrate a similar behavior. On the contrary, benign hosts rarely visit the same IP with the same protocol after we filter out the internal and legitimate flows. It is therefore reasonable to associate all of the flows that follow each triggering flow on a host-by-host basis within a time window. These associated flows are considered action flows initiated by triggering flows. Our analyzer then extracts a set of features from each associated flow group to transform it into a flow feature vector for ease of clustering. There is a possibility that benign hosts visit the same IP with the same protocol. Even so, since their flow patterns are usually different, they cannot form clusters among themselves. We detail this scenario in Section 3.4.4.

Since a flow record is only a brief summary of a session or a connection, the information provided is limited. We make the most of the information by selecting 17 features to constitute a flow feature vector which characterizes not only general traffic patterns but also distinction between benign and malicious hosts at network level. We did so because selecting features essential to all types of botnets can make clustering more effective and accurate, even if our clustering algorithm searches for similarly-behaving hosts and does not require *a priori* knowledge of benign and malicious behaviors. Table 3.2 shows our selections which are mostly statistical features. Features 1 through 14 characterize flow patterns only, which are the sample mean, variance, skewness and kurtosis of flow duration, total bytes transferred, the number of packets transferred, and TCP & UDP break-downs. Features 15 through 17, which are also captured at host level for validation purpose, reveal bots' malicious intent to some degree. Note that benign hosts seldom conduct above activities. Even if a group of benign hosts visit the same destination themselves or the same as bot-infected hosts do, and cannot be filtered out by the trigger-action association, they may be ruled out by our clustering module because their network behaviors are usually different among themselves and different from bot-infected hosts. Compared to bot-infected hosts, benign hosts are less likely to take similar actions after visiting the same IPs because they are not coordinated and commanded to do so.

3.3.2.2 Clustering

Using a vector representation, each associated group of flows becomes a flow feature vector at every time window; this facilitates the task of clustering. Our goal is to group similarly-behaving hosts together by computing the closeness of their feature vectors. In the area of data clustering, two types of algorithms are available: hierarchical and partitional. We use the hierarchical clustering because its clustering result is deterministic and has a structure that is more informative than the result generated by a partitional algorithm. Using the structured result, we can employ a technique to find a good cut of clustering. Specifically, we use the *pvclust* package to calculate *p-values* via multi-scale bootstrap resampling for each cluster in the hierarchical clustering. The p-value of a cluster is a value in [0, 1], indicating how strong the cluster is supported by data. The package provides two types of p-values: AU (Approximately Unbiased) p-value and BP (Bootstrap Probability) value. AU p-value is computed by multi-scale bootstrap resampling, a better approximation to the unbiased p-value than the BP value computed by normal bootstrap resampling [17]. For a cluster with AU p-value greater than 0.95, the hypothesis that "the cluster does not exist" is rejected with a significant level (equal to or less than 0.05). We thus accept a cluster if its AU p-value is greater than 0.95.

3.3.3 Correlation Engine

As described earlier, whenever a group of hosts is identified by the clustering module as a cluster, the respective host analyzers are required to report the suspicion levels along with network statistics to the correlation engine, since the results generated by flow analysis alone may not be accurate and further in-host validation is needed. Given the two sources of information as input, the correlation engine produces a detection result for each host.

Based on the consistency check of network statistics, there are two possibilities. First, the network features sent from a host are falsified and differ from those observed at the network level. In such a case, the correlation engine considers the host compromised and generates the detection result immediately. Another possibility is that the network-level results are consistent, then we need to consider both the in-host suspicion-level and the quality of the clustering. The detection result should be a function of these two parameters. It is straightforward that the higher the suspicion level the more likely a host is part of a botnet. To quantify the contribution of the clustering quality, we need a measure to reflect the closeness of each host to its clustered group. In other words, the more similar a host's network behavior is to other hosts in the same cluster the more likely it is part of a similarly-behaving botnet. This measure can be the average distance from a specific host to other hosts. We used the "correlation" method to gauge the distance. We do not use other distance measures because the correlation values in our data set are mostly positive. A study [42] has shown that in this scenario, the "correlation" method performs best.

Assume that a cluster consists of n hosts each of which is represented by a 17-dimensional flow feature vector, forming a $17 \times n$ matrix $X = \{x_{ij}\}$. The i-th row corresponds to the i-th feature of these hosts and the j-th column corresponds to a flow feature vector. The distance between host u and host v is given by

$$D_{uv} = 1 - \frac{\sum_{k=1}^{17}(x_{ku} - \overline{x_u})(x_{kv} - \overline{x_v})}{\sqrt{\sum_{k=1}^{17}(x_{ku} - \overline{x_u})^2}\sqrt{\sum_{k=1}^{17}(x_{kv} - \overline{x_v})^2}}.$$

where $\overline{x}_u = \frac{1}{17}\sum_{k=1}^{17} x_{ku}$ and $\overline{x}_v = \frac{1}{17}\sum_{k=1}^{17} x_{kv}$. As the correlation is always in the range of [-1,1], D_{uv} belongs to [0,2] and so does the average distance $\overline{D_n}$.

Now, we have two parameters in the correlation algorithm. One is the suspicion level S_n, and the other is the average distance $\overline{D_n}$. The final detection score is denoted by $Score_n$ and given by

$$Score_n = w_1 * S_n + w_2 * f(\overline{D_n}). \tag{3.2}$$

f is a function that maps each average distance $\overline{D_n}$ to a value in [0,1], having the same range as that of S_n. w_1 and w_2 are weight factors. Recall that the smaller the average distance, the more similarly-behaving a host to other hosts in the cluster and the more likely it is part of a botnet. To reflect this concept, we selected a decreasing function $f(\overline{D_n}) = 1 - \frac{\overline{D_n}}{2}$. Since at the beginning we cannot completely trust the host-level information, we assign w_1 to 0.1 and w_2 to 0.9, making our detection rely more on the network-level analysis, which is especially important when a host analyzer is compromised. Every time the network feature consistency check passes, w_1 increases by 0.05 and w_2 decreases by 0.05 until they reach 0.5. The final detection $Score_n$ is a value in [0,1].

Table 3.3: Botnet traces

Trace	Duration	Number of Bots
IRC-rbot	24h	4
IRC-spybot	32m	4
HTTP-BobaxA	4h	4
HTTP-BobaxB	20h	4
Storm	48h	4
Waledac	24h	4

3.4 Evaluation

3.4.1 Data Collection

We have evaluated the performance of our framework in detecting 3 types of botnets with real-world traces—IRC-based, HTTP-based, and hybrid-P2P. We set up VMWare virtual machines running Windows XP, connected via a virtual network to monitor and collect traces. While running these botnets, we also ran a variety of benign applications at the same time to make these machines behave similarly to real compromised hosts. Both the benign and malicious behaviors at the Registry, file system, and network stack were captured. Table 3.3 shows the details of these botnet traces, each containing 4 bot instances. The modified source code of IRC-rbot and IRC-spybot were used in the virtual network to generate their respective traces. We obtained the binaries of HTTP-based BobaxA and BobaxB, and hybrid-P2P-based Storm and Waledac from public web sites. The IRC- and HTTP-based botnets' network-level traces were captured within a controlled environment and transformed from packet data to flow data in our experiment. Since Storm and Waledac botnets were still active in the wild at the time when we collected data, we carefully configured the firewall setting and connected virtual machines to the external network so that the bots actually joined the real Storm and Waledac botnets, and our campus router captured all of the bots' traffic.

We also collected 5-day NetFlow data from a campus network core router which covered the flows generated by Storm and Waledac instances and all other hosts in the network. Our campus network administrator confirmed that all hosts in the 5-day flow data except for those running Storm and Waledac were benign, meaning that there was no botnet traffic present in other hosts during that period. Thus, it is valid to assume that these hosts are benign at both host- and network-level (other types of malware might run on these hosts but they are not our detection targets). The 5-day data consist of three sets: (1) 2-day data containing 48-hour Storm traces; (2) 1-day data including Waledac; and (3) other 2-day data. We divided the third data into two subsets, 1-day each. Note that overlaying malicious traffic on clean traffic for evaluation has been commonly used in malware detection literature [44, 45, 96]. Although our botnet traces already contained benign traffic, the amount of such traffic was limited and we wanted to add more to make it more realistic. Thus, we overlaid the botnet network traces except Storm and Waledac, one at a time, on data set (3), two traces on the first day, and two on the second day. For example, the IRC-rbot included 4 bot instances, and we randomly selected 4 hosts from the clean 1-day traffic and replaced the bots' IPs with the selected IPs. We treated Storm traces in the same way and intentionally overlaid the 1-day Waledac traffic on HTTP-intensive benign hosts, the purpose of which will be described later. In addition, hosts running P2P clients are important for the evaluation of our detection framework as one may wonder if they will be misclassified as bots. Since NetFlow data could not reliably identify which hosts had P2P activities, we ran P2P applications such as eMule and BitTorrent on hosts under our control and collected their host- and network-level traces. We obtained 4 sets of hour-long traces from hosts running eMule and 3 sets from those running BitTorrent. While conducting P2P activities, these hosts also ran other regular network-relevant applications, such as web-browsing, ssh and email-checking. In what follows, we will show the overhead of the system, the detection accuracy, and the benefit of combining both host- and network-level information.

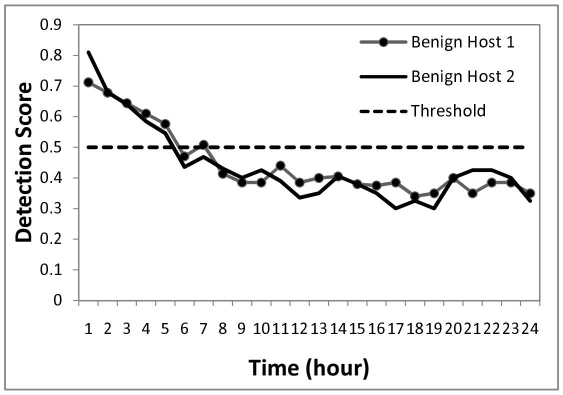


Figure 3.2: The change of detection scores for two benign hosts

3.4.2 Overhead

One may want to know the overhead incurred by the three components of our framework. To measure the overhead of the host analyzer, we used a common Windows benchmark PassMark Software, PerformanceTest [14]. Our host analyzer was implemented on a machine with AMD Athlon 64 3200+ Processor 2.0GHz, 1GB of memory, 80GB of disk, and Windows XP operating system. We ran the benchmark program for CPU, memory and disk, respectively, 5 rounds each. The average overhead for CPU is 3.1%, memory 3.5% and disk 4.7%. The in-host suspicion-level generator can determine one host's suspicion level in about 10 μs given the behavior vector. Since the SVM is pre-trained (i.e., the support vectors are pre-loaded), the training process will not incur any runtime overhead to the host. Our network analyzer and correlation engine were implemented in Linux kernel 2.6.18 on an HP ServerBlade with 2 Dual-Core AMD Opteron (tm) Processors 2.2 GHz, 4 GB of RAM, and 260 GB of disk space. The network analyzer can parse 1-hour flow data and cluster similarly-behaving hosts within 2 minutes on average. To assign the final suspicion score and produce a detection result, the correlation engine spends 1 second per host on average.

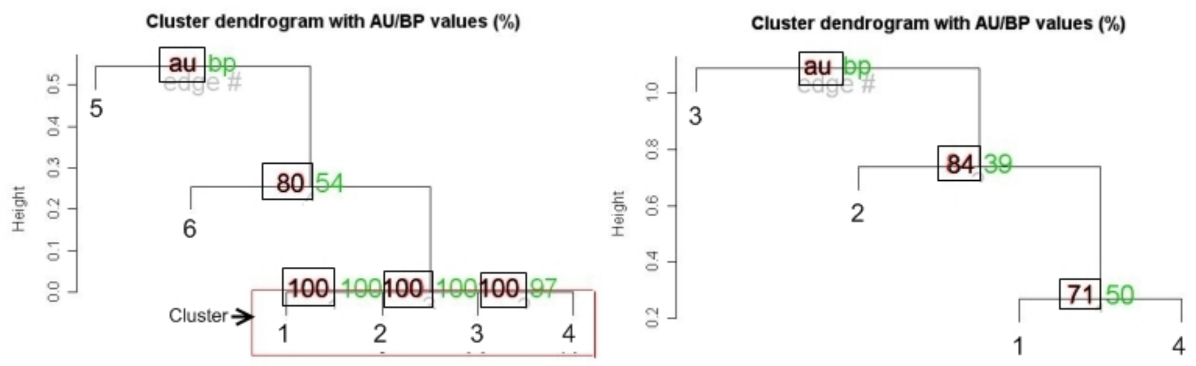


Figure 3.3: Cluster dendrograms with AU/BP values (%) (Left graph: clustering of bots and benign hosts; Right graph: clustering of benign hosts)

3.4.3 Detection Results

We now report the detection results on 6 botnets. The performance of our detection framework was measured by false-alarm rates, i.e., false-positive (FP) and false-negative (FN) rates. A false-positive is defined as a benign host mistakenly classified as a bot-infected, and a false-negative means that an actual bot-infected host fails to be detected. Recall that the detection score is in the interval [0,1]. The detection threshold was set to 0.5 in our evaluation to strike a balance between FP and FN rates, and this parameter is configurable. There is always a tradeoff between FP and FN rates. A lower threshold can be set if FNs are a concern, while a higher threshold is required if FPs are less desirable.

Table 3.4 shows our evaluation results where the average number of FP or FN hosts is calculated during the entire period of evaluation. The average FP or FN rate is the number of FP hosts divided by the total number of benign hosts (around 2,000), or the number of FN hosts divided by the total number of bot-infected hosts. Our framework was able to identify almost all bot-infected hosts. There was only one bot undetected, generating a false-negative.

Our framework also performs well in terms of false-positives. The highest false-

Table 3.4: False alarm rates

Trace	Avg FP hosts	Avg FP	Avg FN Hosts	Avg FN	Duration
IRC-rbot	3.208	0.0016	0.125	0.0313	24h
IRC-spybot	2.833	0.0014	0	0	24h
HTTP-BobaxA	1.000	0.0005	0	0	24h
HTTP-BobaxB	1.083	0.0005	0	0	24h
Storm	2.563	0.0013	0	0	48h
Waledac	0.9167	0.0005	0	0	24h

positive rate was no greater than 0.16%. It turned out that almost all false-positive hosts appeared during the first few hours of the traces due to the values of "untuned" weight factors w_1 and w_2. As mentioned before, we set w_1 (the weight of suspicion level) to 0.1 and w_2 (the weight of clustering quality) to 0.9 at the beginning to reflect lack of confidence in the host-level information. During the first few hours our framework relied more on the network-level analysis, resulting in detection inaccuracy when a group of benign hosts demonstrated similar network behaviors among themselves (e.g. they ran the same network applications) or behaved similarly to bot-infected hosts. As the host-level information was verified to be trustable, w_1 increased and w_2 decreased so that host-level information gradually had a higher weight and was able to correct the detection results. Figure 3.2 shows the change of the detection scores on two benign hosts which have similar network traffic patterns and form a cluster by themselves. At the beginning, both of them have greater than 0.5 detection scores due to the high weight assigned to the clustering quality parameter, leading to false-positives. As time goes by, the suspicion-level parameter receives a more balanced weight. Since their suspicion levels are always low (0 to 0.1), their final detection scores decrease below the 0.5 threshold and no longer incur false-positives. Network- and host-level information indeed complement each other, and hence combining them while making a detection decision is the key to reducing false alarm rates.

3.4.4 Evaluation with Network Analyzer

Using the network analyzer that performs flow analysis and clustering, we found some interesting results. The trigger-action association done by the flow analyzer can significantly narrow the number of hosts for clustering because benign hosts rarely visit the same IP with the same protocol after traffic filtering, while bot-infected hosts connect to the same group of C&C servers or peers. Even if benign hosts cannot be filtered out by trigger-action association, they are likely to be discarded by the clustering module because their flow patterns are usually different among themselves and different from bot-infected hosts. This fact makes the clustering module effective in reducing the number of benign hosts appearing in the final clusters.

Figure 3.3 shows the hierarchical clustering dendrogram of scenarios in which a few benign hosts were ruled out not by the trigger-action association but by the clustering module. The graph on the left is the scenario when bot-infected and benign hosts happened to visit the same destination and their flow feature vectors were sent to the clustering module for grouping. There are 6 hosts to be clustered, numbered from 1 to 6. 1 to 4 are bot-infected hosts, and 5 to 6 are benign hosts. Recall that we use hierarchical clustering with AU p-values indicating how strong the clustering is supported by data. Normally, clusters with p-values greater than 95% are considered reasonable clusters. The AU p-values and reasonable clusters are highlighted by rectangles in the figure. In the left graph, 4 bot-infected hosts are clustered together with 100% AU values, meaning that their flow feature vectors are quite similar. The two benign hosts in the graph cannot form a cluster with them because of the dissimilarity in flow patterns between the benign and bot-infected hosts. The graph on the right represents the scenario when a few benign hosts visited the same destination. 4 hosts, numbered from 1 to 4, are all benign. The 4 benign hosts cannot make any cluster (low AU p-values), since their flow feature vectors differ significantly.

We also collected additional data from several benign hosts running P2P applications to see if false-positives would occur. Four hosts each ran an eMule client, and three other hosts each ran a BitTorrent client named *utorrent*. Besides P2P file sharing, these hosts also made other network accesses during the period of trace collection. This is realistic because normally a P2P user does multi-tasking during file-sharing, rather than solely waiting for the file-sharing to complete. We used the network analyzer to perform flow analysis and clustering on these P2P data sets. It turned out that the four eMule hosts did visit the same IPs (servers) so that they were not ruled out by the trigger-action association and needed to be clustered. The same thing happened to the three utorrent clients. However, during the clustering, those P2P hosts could not make any cluster. We found that the AU p-values generated for the four eMule hosts were no greater than 85% and for the three utorrent ones no greater than 90%, both of which were below the 95% clustering threshold. That is, these benign hosts did not behave similarly at the network level even though they ran the same P2P client. One reason for this is that P2P file-sharing is a user-specific activity. Users have different interests and download or upload different files so that the flow features, such as total bytes, number of packets and number of TCP or UDP flows are hardly similar. The other reason is that network activities other than P2P also add some dissimilarity to the flow patterns among hosts running P2P applications. Although in our experiment, P2P hosts were ruled out by the clustering module, we still inspected their host-level behaviors to make sure that even if the network analyzer failed to distinguish them, the host analyzer could tell they were benign. The results were in line with our expectation: the suspicion-levels for these hosts were always much less than 0.5, because there was little malicious behavior demonstrated at the host level. In this scenario, since the correlation engine considers both types of information, it will generate correct detection results with the help of suspicion-levels even if the network analyzer cannot rule them out.

In summary, our network analyzer—the flow analyzer along with the clustering module—is effective in forming suspicious clusters, but it may fail in some situations in which the host analyzer needs to assist. We present such a case study next.

3.4.5 Evaluation with Host Analyzer: A Case Study of Waledac

Waledac worm spreads as an attachment to a spam email or through a link to a malicious website. It came into the wild at the end of 2008 and has not yet been completely taken down (as of March 2010). The Waledac botnet uses the HTTP protocol for C&C traffic forwarding and the botmasters are well hidden behind a P2P network [40]. We downloaded samples by following Waledac spam's links to its malicious domains in Feb 2009.

In our evaluation, we intentionally overlaid Waledac's network traces on benign hosts with heavy HTTP traffic. We did this because Waledac appeared stealthy in its network activities, and we wanted to evaluate how well our framework can perform in the situations where the network analyzer cannot distinguish between benign and bot-infected hosts. We observed that these Waledac instances did not send any spam email in the 24-hour period. The only activity was several HTTP sessions every hour for C&C such as transferring locally-collected information. This type of malicious traffic is easy to blend into benign HTTP flows but hard to isolate.

Over a few time windows, our network analyzer mistakenly clustered one benign host into the same group as 4 Waledac bots. One reason lies in the way we mixed benign and bot's traffic for bot-infected hosts (we did this intentionally). It turned out that the HTTP-intensive benign host and the 4 bot-infected hosts had visited the same destination IP using the HTTP protocol. As shown before, only visiting the same IP is not enough for forming a cluster. To be grouped into the same cluster, the hosts should have similar traffic patterns. For most bot-infected hosts, although their flows are mixed with benign flows, their malicious flow patterns may still be conspicuous because of their distinct and aggressive spam sending and scanning activities. In other words, their network-level behaviors can be distinguished from those of benign hosts during the clustering. However, in the Waledac's case, the stealthy C&C traffic was hidden and diluted into the benign HTTP traffic, and the clustering module failed to differentiate bot-infected hosts' network activities from those of the benign hosts, which is the other reason for the incorrect clustering. Nevertheless, our framework still correctly generated the detection results for Waledac bot instances, thanks to the information obtained by the host analyzer. As the Waledac exhibited malicious behavior at the host level, each bot-infected host's suspicion level was 0.88 on average. On the other hand, the benign host's suspicion level was close to 0. The final detection $Score_n$ for bot-infected hosts was as high as 0.85, while that for benign hosts was 0.40. Without the host analyzer, benign hosts are likely to be misclassified as bot-infected ones in the presence of bots that are stealthy at the network level. In other words, relying solely on the network-level analysis cannot create a complete picture: the inspection of in-host behavior by the host analyzer is critical in reducing false-positives.

3.5 Discussions

One limitation of our approach is its scalability since the approach requires runtime host-level analyzers. Our design is intended for use in edge networks such as enterprise networks where a security framework can be enforced easily on all hosts. In large-scale networks, if host analyzers cannot be installed on every host, we may use available host analyzers to infer the suspicion levels at those without the analyzers if they form the same suspicious cluster by network-level analysis.

Since our network analyzer looks for trigger-action patterns among hosts, bots may delay their coordinated actions by waiting for random period of time. To counter this evasion, we may lengthen our time window of analysis or randomly select a time window that cannot be figured out by attackers. As the goal of a botnet is to perform malicious actions, if each bot does not act maliciously for a very long time, it will be ineffective, causing few problems. Bots may also attempt to randomize their traffic patterns such as injecting random number of packets in each flow or they can mimic flow patterns of benign hosts. However, these techniques may not help bots much to evade our detection because our framework also considers host-level behavior while making detection decisions.

Since our framework is deployed in a monitored network, hosts within the network are geographically close to one another. It is natural for bots to connect to, or bootstrap from, the same set of nearby IPs to receive commands and take actions in a coordinated manner. To intentionally evade our network analyzer, bots may use different C&C servers or contact a different set of IPs. If there are a large number of bots in our network, our approach may group them into several suspicious clusters. However, if there is only one or a few bots (without contacting the same set of IPs), our detection framework should obtain information from the host analyzers more frequently. If a host's suspicion level is high enough, the detection result can be generated even without any suspicious cluster.

Another possible evasion of our framework is to compromise the host analyzers and send falsified information to the correlation engine. This can happen only if the bot sits below our host monitoring level and is able to modify or subvert the system-wide information the in-host monitor receives. Our current solution is to gather a few network statistics from the host to compare against those observed by the network analyzer. A bot may keep the network statistics intact and modify the suspicion-level information only to mislead the correlation engine. Even if this happens, there is still a high possibility of capturing the bot, because the weight factor assigned to the host-level information by the correlation engine is much lower than the weight factor assigned to the network-level information (i.e., clustering quality) at the beginning.

It means that with a high weight factor on network information, as long as the compromised host exhibits correlated malicious network-level activities and forms a suspicious cluster with other hosts, it will be detected with high probability despite the falsified suspicion-level from the host analyzer. To further counter this attack, we may use secure hardware or secure VMM to safeguard the OS as well as our monitors in each host.

While it is likely that our framework could be evaded, the evaluation results demonstrated that our system performed well in detecting state-of-the-art botnets with minimal impact on benign hosts. Therefore, our system raises the bar against botnets.

3.6 Conclusion

Considering the coordination of bots within a botnet and each bot's malicious behavior at the host level, we proposed a C&C protocol-independent botnet detection framework that combines both host- and network-level information. Our evaluation based on real-world data has shown the following results. The network analyzer is effective in forming suspicious clusters of aggressive bots but fails to separate benign hosts from bot-infected hosts if the latter are stealthy at the network level. When the stealthy bots are present, it is the host analyzer that provides correct detection results by generating distinguishing suspicion levels. By using combined host- and network-level information, our framework is able to detect different types of botnets with low false-positive and false-negative rates.

CHAPTER IV

Large-Scale Botnet Detection at the Internet Infrastructure

4.1 Introduction

A botnet consists of a group of coordinated bots that can mount attacks such as Distributed Denial of Service (DDoS), spamming, phishing and identity theft. Botnets are posing a serious security threat to the Internet users; they can bring down the entire system and disrupt Internet services. In a botnet, a Command and Control (C&C) channel, in which a botmaster disseminates commands to, and get response from bots, is a key element. Attackers have recently devised a decentralized C&C infrastructure exploiting the P2P protocol. A few noteworthy P2P botnets in recent years include Storm [22], Waledac [25] and Conficker [6]. Their P2P implementations are either based on an existing protocol (Storm utilized Kademila [61]) or completely customized.

The decentralized nature of P2P botnets inevitably challenges detection attempts. Approaches targeting centralized C&C structures [29, 43, 45, 56] become ineffective under the new structure in which a botmaster can join, issue commands and leave at any time at any place. Generic detection approaches [44, 98] relying on behavior monitoring and traffic correlation analysis are mostly applicable at a small scale such as in edge networks and do not scale well because they require analyzing vast amounts of fine-grained information. In addition, if there is only a small number of bots in an edge network, detection based on bots' coordination may fail due to the limited number of instances in view. Given the fact that current botnets' sizes are in the order of hundreds of thousands, an effective and efficient large-scale detection needs to function at a high level without requiring fine-grained information that can only be obtained locally. As a P2P botnet has a structured overlay and connectivity patterns different from other applications from a graph analysis perspective, naturally, we consider detection at the Internet infrastructure level by assessing the impact imposed by a P2P botnet at various network components and measuring the effectiveness of detection at such places.

4.1.1 Contributions

In this chapter, we evaluate the feasibility of detecting large-scale P2P botnets with different network components at the Internet infrastructure level. We construct three types of P2P-botnet overlays, map them to the corresponding AS (Autonomous System)-level underlays by inferring each overlay connection's AS-path, and accordingly determine the PoP (Point of Presence) path and geographical router rendezvous (co-located routers in the infrastructure) each connection goes through. We then take a close look at each individual AS, PoP and router rendezvous based on graph analysis. In particular, we calculate a few P2P traffic classification metrics to see whether the portion of botnet connections observed by a single network component can be identified as P2P traffic. We would like to answer the following three questions through our analysis: (1) which network component is the best place for detection? (2) which P2P overlay structure can help hide the botnet traffic well? (3) what are the limitations of detection at the infrastructure level? Our main contribution lies in the thorough analysis of detection potential at the three infrastructure-level network components for three different P2P overlay topologies.

Our analysis has led to three key observations. First, a small number of ASes can observe almost all overlay connections, but the AS-level detection is less practical. PoPs can capture a large fraction of connections but the number of monitoring points is limited. Router rendezvous strike a balance between detection capability and feasibility. Second, a botnet has to make a tradeoff between resilience/efficiency and the ability to evade detection. Third, the infrastructure-level detection is not a panacea for all large-scale botnets: it needs to be integrated with detection schemes in edge networks to complete a detection picture.

4.1.2 Organization

The remainder of the chapter is structured as follows. Section 5.2 describes related work. Section 4.3 details our methodology. Section 4.4 presents analysis results. Section 4.5 discusses a few challenges associated with our approach. The chapter concludes with Section 5.6.

4.2 Related Work

Considering the fact that P2P botnets have structured overlay topologies, our approach takes a global view, exploiting structural properties derived from graph analysis and is thus not limited by the availability of fine-grained information. In this regard, our work is closely related to graph-based traffic classification and analysis. Iliofotou *et al.* [53] proposed the use of Traffic Dispersion Graphs (TDGs) to monitor, analyze, and visualize network traffic. TDGs focus on network-wide interactions among hosts and show that graph features, such as the average degree and directionality, can be utilized to distinguish different applications. Using TDGs, they further classified P2P traffic at the Internet backbone [52]. Their scheme filters out known traffic, forms traffic clusters roughly based on applications, and finally, uses some graph metrics to identify whether a cluster belongs to a P2P application. In our analysis, we adopt some of their metrics to determine whether the portion of traffic observed by a network component is P2P. BotGrep [65] analyzes structured graph to locate bots by extracting P2P subgraphs from a communication graph containing background traffic. This approach assumes the visibility of the entire botnet communication graph, whereas our detection is at a single network component where only a fraction of botnet communication can be seen.

We are aware of two published results on AS-level underlays mapped from P2P overlays. Rasti *et al.* [73] examined the global impact of the load imposed by a P2P overlay on the AS-level underlay. They use Gnutella network snapshots to analyze diversity and load on individual AS-paths, churn among the top transit ASes and propagation of traffic within the AS-level hierarchy. Their focus was on the effect of overlay on the underlay, while our work is concerned with whether the effect can be utilized for detection. Jelasity *et al.* [54] constructed a modified Chord [82] topology and showed that the visibility of P2P botnet traffic at any single AS is limited and not sufficient for detection. Our method differs from theirs in the following aspects. First, we consider bots' geographical distribution in the overlay topology while they assume that the number of overlay nodes in each AS is proportional to the size of the AS. Second, our AS-level paths are not derived from the shortest-path algorithm they used, but a more realistic scheme. Third, we simulate three P2P overlay topologies and observe the traffic not only at the AS-level but also at PoPs and router rendezvous, providing a more thorough analysis.

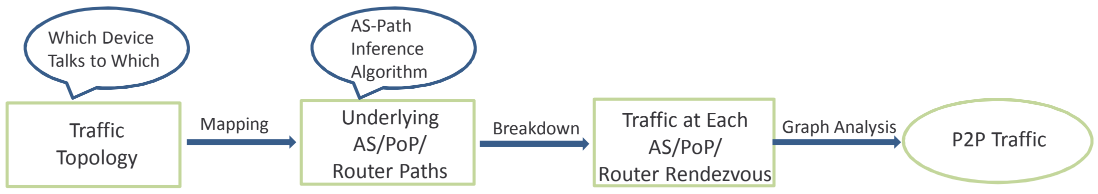

Which Device Talks to Which
AS-Path Inference Algorithm
Traffic Topology
Mapping
Underlying AS/PoP/ Router Paths
Breakdown
Traffic at Each AS/PoP/ Router Rendezvous
Graph Analysis
P2P Traffic

Figure 4.1: Overview

4.3 Methodology

4.3.1 Overview

We would like to achieve the following two goals. First, from a defender's perspective, we would like to see how much of the botnet connections can be observed at a single network component and whether the respective communication graph has P2P properties. Second, from an attacker's perspective, we want to study which P2P overlay topology is stealthy enough so that at a single network component the graph-level information is insufficient for detection. Our methodology consists of four main steps as shown in Figure 4.1. In the first step, we construct a P2P overlay topology based on simulation and learn which end-device talks to which, i.e., the overlay connections. In the second step, to map the overlay to the AS-level underlay, we associate a connection's two end-devices' IP addresses with the corresponding ASes and calculate the AS-level path between the two ASes. Given the AS paths, we then determine PoP-level paths and geographical router rendezvous paths. Knowing the paths of all connections, in the third step, we break down the connections on per-AS, per-PoP, and per-router-rendezvous bases. We are especially interested in the top ASes, PoPs and router rendezvous ranked by the number of connections going through. In the last step, we inspect those top network components individually. As in [54, 73], we do not consider background traffic but focus only on the traffic coming from the P2P overlay, which is the best scenario. It implies that if the P2P traffic cannot be identified under this situation, it will definitely not be captured when background traffic

Table 4.1: Data sources used in our Internet infrastructure and end-device model

Model component	Data sources
Backbone topology	Skitter dataset: http://www.caida.org/tools/measurement/skitter/ Alias clustering data from the iPlane project: http://iplane.cs.washington.edu/data/alias_lists.txt IP geolocation dataset: http://www.ip2location.com/
Internet Point of Presence	Telegeography co-location database: http://www.telegeography.com/
Internet end-devices	US census data: census-block population in each $250 \times 250 m^2$ grid in the US for a 24-hour duration [62]
Internet access routers	Dial-up service aggregators per each zip code: http://www.findanisp.com Broadband ISP market share: http://www.leichtmanresearch.com/press/081108release.html DSL central office locations: the LERG (Local Exchange Routing Guide) dataset from Telcordia Cable company service locations: Dun & Bradstreet (D&B) dataset
Internet routing	BGP routing information from the University of Oregon Route Views Project: http://www.routeviews.org/ AS prefix sets: http://www.fixedorbit.com/ AS-level path inference: Qiu and Gao's algorithm [71]

is present. We analyze several graph properties of the communication patterns at each top network component and determine whether it has the characteristics of P2P traffic.

4.3.2 Internet Infrastructure and End-Device Modeling

Before detailing the four main steps, we would like to briefly describe the Internet infrastructure and end-device modeling, which lays a basis for our methodology. We use multiple real-world datasets to construct a realistic model of the US Internet infrastructure. Table 4.1 lists all data sources in the model construction. In total, 73,884,296 residential computers are generated in the entire US (except Hawaii and Alaska). The distribution of Internet access routers including dial-up, DSL and Cable is based on the market share of top US broadband companies and dial-up service aggregators, and how these access routers connect to the backbone topology at Internet PoP locations is derived from AS peering relationships. We refer interested readers to [95] for details of this modeling.

4.3.3 Overlay Topology Construction

In recent years, P2P overlays have become popular in botnet construction due to their decentralized nature. Many existing P2P overlays can be utilized to facilitate botnets' C&C. We construct three types of P2P overlays: a widely-used Kademlia [61], a modified Chord [82] and a simple ring structure. We will later compare the structural properties of these three overlay topologies at each network component, the results of which will be presented in Section 4.4. Next, we will briefly introduce each P2P overlay followed by the way we construct the topology.

4.3.3.1 Kademlia

Kademlia is a Distributed-Hash-Table (DHT)-based P2P overlay protocol. Under this protocol, there is no central server and resource locations are stored throughout the network. Nodes are identified by node IDs and data items are identified by keys generated from a hash function; node IDs and keys are of the same length. Data items are stored in nodes whose IDs are close to data items' keys. The distance between two IDs, X and Y, is calculated by bitwise exclusive or (XOR) operation: $X \oplus Y$. To search a data item, a node queries its neighbors for nodes whose IDs are close to this data item's key. After getting responses from its neighbors, the node continues to query those nodes that are closer to the key. This iterative process repeats until no closer nodes can be found. The benefit of Kademlia is its resilience to disruptions. Even if a few nodes are shut down or removed, the network will still be able to function. Kad network is an implementation of Kademlia. A few major P2P file sharing networks adopt the Kad implementation, such as Overnet and eMule. The Storm botnet was built upon Overnet.

An ideal way to construct the botnet overlay topology is to collect traffic traces from a real network, such as the Storm botnet. Since the Storm botnet is decentralized (i.e., there are no central venues where all communications can be observed), traces captured from the Storm botnet fall into two categories each of which has its drawbacks. In the first category, the traffic data were collected from a single or a few vantage points. They can only provide partial views of the entire botnet. In the second category, snapshots of the network were taken by network crawlers. The snapshots contain information, such as which IPs are alive or dead but cannot tell which IP connects to which IP. To characterize the effectiveness of detection at the underlay, a full picture capturing the entire network's connections is indispensable, so we have to construct a Kad network by simulation.

We use a high-fidelity botnet simulator BotSim [46] which integrates a popular P2P client named *aMule* [1], an implementation of Kad. Considering the fact that simulating a large-scale botnet (100,000 bots) on a single or a few machines will take a prohibitively long time, our simulator was run on a distributed platform consisting of 400 machines, each with 2 Pentium III CPUs and 4Gb RAM. The simulator is a component of MIITS [91] which is built upon PRIME SSF [15], a distributed simulation engine utilizing conservative synchronization techniques. To make aMule work seamlessly on our simulator, several modifications were made to the original aMule code including intercepting time-related system calls and substituting them for simulated time function calls, and replacing socket API calls with network functions developed in MIITS. The rest of the code remains intact.

In a botnet, a majority of bots are compromised residential computers and not necessarily geographically close, and hence we have to take locations into account. Constrained by data availability, all bots in our simulation are in the US and their locations follow the geographical distribution of 73 million residential computers by state. The simulation of 100,000 bots executes for three days in simulation time. The output files log timestamps and connections in the network. We discarded the first day in which bots bootstrap and the entire botnet stabilizes, and kept the second and the third day for analysis. With log files keeping track of which node talks to which other node and each node's state information, we need to obtain the IP address of each end-device to completely construct the overlay topology. For this, we randomly chose an end-device address from the state a bot resides in. This way, we created two Kad overlay topologies with 100,000 nodes, one day each.

4.3.3.2 Modified Chord

Chord is a DHT-based P2P protocol under which nodes form a ring structure. Each node has a predecessor and a successor and a few long range links. For example, there are a total of N nodes in the ring. Node i connects to nodes $(i-1) \mod N$ and $(i+1) \mod N$. It also connects to nodes $(i+2^k) \mod N$ for $k = 1, 2, \ldots, log_2 N - 1$ to form long-range links. In [54], modifications to Chord are proposed so that it is difficult to detect the botnet through graph analysis at any single AS. The main modification is creating clusters in the ring each of which has $\log_2 N$ consecutive nodes. This way, nodes in the same cluster can share the same set of long-range links for routing. This topology is of interest to us because we want to see whether using a more realistic AS-path calculation algorithm can make a difference in detection and whether this topology can successfully hide itself at PoPs and router rendezvous as well. Since this modified Chord's topology is relatively simple, we constructed its overlay with 100,000 nodes directly based on its protocol without simulation. Following the same practice as in Kademlia, each end-device address is a random draw from the state a bot belongs to.

4.3.3.3 Simple Ring

We also consider the simplest case: each node has only two neighbors—a predecessor and a successor—to construct a ring structure. Presumably, this structure is stealthier and harder to detect than the modified Chord due to lack lack of connectivity at the overlay. We will verify this presumption in later analysis. Similar to the modified chord, this overlay has 100,000 nodes constructed directly and the bots' locations follow the same geographical distribution.

4.3.4 Overlay to Underlay Mapping

4.3.4.1 AS-Path

Given all overlay connections, the next step is to map each connection to an AS-level path. Note that each end-device IP address is associated with an AS number and determining an AS-path of a connection is actually to determine the AS-path between two ASes. We use the AS-path inference algorithm in [71] for inter-domain routing. The key idea is to infer AS paths from existing BGP routing tables.

4.3.4.2 PoP-Path

A PoP is an access point to the Internet. It is a physical location owned by an ISP or located at Internet exchange points and co-location centers. The computation of a PoP-level path is based on the respective AS-level path. Given a pair of source and destination end-device IPs, the algorithm first determines the AS-level path $AS_1AS_2 \ldots AS_n$, then iteratively finds the shortest IP-level path between PoPs connecting every neighboring pair of ASes and finally maps the IP-level path to the PoP-level path. We refer interested readers to [95] for details of this algorithm.

4.3.4.3 Router Rendezvous Path

Given an IP-level path of a connection, the geographical router rendezvous along this particular path can be determined directly.

4.3.5 Traffic Breakdown

Since our work focuses on structural properties of traffic graph at a single network component (AS, PoP or router rendezvous), not the entire botnet overlay per se, we need to break the traffic down on a per AS, per PoP and per router rendezvous basis. This breakdown process is straightforward. We then rank the three types of network components by the number of connections going through, and take a close look at the graph properties observed at each of the top 10 ASes, PoPs, and router rendezvous, respectively, in our analysis.

4.3.6 Graph Analysis

After breaking down the traffic, we know all connections that traverse a particular AS, PoP and router rendezvous. We can then generate directed graphs in which bots are represented by vertices and connections among them are represented by edges. For simplicity, all edges carry the same weight. Graph metrics to determine whether the traffic is P2P are proposed in [52] and adopted to analyze the modified chord in [54]. In our analysis, we inspect the same set of features as in [54] for consistency. The features used to characterize P2P traffic include the number of weakly-connected components, size of the largest weakly-connected component, average node degree and InO (In Out) ratio. We introduce each of them as follows.

Number of Weakly-Connected Components: A weakly-connected component is a maximal subgraph of a directed graph such that in the subgraph replacing all of its directed edges with undirected edges produces a connected undirected graph. For effective detection, we expect a small number of weakly-connected components. As one can imagine, a large number of connected components usually means small-size components that are less likely to exhibit typical P2P patterns.

Size of the Largest Weakly-Connected Component: This metric is meaningful to us because as pointed in [53] the graph formed by a P2P network tends to be densely connected and have a large connected component including the majority of participating nodes.

Average Node Degree: This metric counts both the incoming and outgoing edges of a node, i.e., ignoring the directionality. A graph with a high average degree tends to be tightly-connected and P2P networks normally have high average node degrees.

InO Ratio: The metric calculates the percentage of nodes in the graph that have both incoming and outgoing edges. This metric is of interest because under client-server protocols such as HTTP and SMTP, clients usually initiate connections (outgoing edges) whereas servers normally accept connections (incoming edges). But nodes in P2P networks usually serve as both clients and servers so that P2P's InO is distinctively higher than others.

4.4 Analysis Results

This section presents our analysis results. Recall that we construct three different P2P overlay topologies, namely, Kad, the modified Chord and the simple ring, and examine their traffic graphs, respectively, at three types of network components. As introduced in Section 4.3.6, the graph features characterizing P2P patterns are the number of weakly-connected components, size of the largest weakly-connected component, average node degree and InO ratio. We conduct graph analysis first at the AS-level, then the PoP-level and finally, the router-rendezvous-level, and show the graph features at the top 10 places of each level.

4.4.1 AS-Level Analysis

We first take a look at the AS-level graphs of three different topologies. Table 4.2 shows the Kad graph properties for day1 and day2, respectively, at top 10 ASes, ranked by the number of unique connections going through. We map the AS numbers to ISPs using the AS-name lookup list [3]. It turns out that from day1 to day2 the top 10 order changes slightly but the 10 AS numbers remain the same. As expected, these top ASes belong to large ISPs such as AT&T and Verizon. Note that the traffic percentage at a single AS is calculated by the number of unique connections observed at that particular AS divided by the total number of unique connections in the entire overlay topology. Since one connection usually can be seen at more than on AS (this is why the third column of the table adds up to more than 100%), we count each connection only once while calculating the number of connections observed at multiple ASes altogether. Following such calculations, in both days, top 10 ASes aggregated together can observe 98.95%—almost all of the Kad overlay's unique connections. In particular, the top 1 AS (3356/Level3) alone can see two thirds of the overlay traffic with all nodes (100000) in the picture in both days. Even for ASes carrying fewer connections, they have at least 99912 nodes' connections traverse through. Most importantly, at each top AS, all nodes are weakly-connected with each other, forming one giant weakly-connected component. This property can facilitate detection because one single weakly-connected graph containing a majority of connections is more likely to demonstrate P2P characteristics and easier to get caught than a disconnected graph with many connected components of small sizes. As suggested in [52], two metrics can characterize P2P traffic. One is a high average degree (larger than 2.8), and the other is a high InO ratio (large than 1%). In both days, at all top ASes, the average degrees and InO values are high enough for P2P classification: the lowest value of average degree is 56.8 and that of InO is 87.75%. Thus, as we can see, all top AS venues have high visibility of Kad's overlay which demonstrates typical P2P patterns, sufficient for detection.

Table 4.3 presents graph features of the modified Chord at top 10 ASes. Compared to Kad, top 10 AS numbers remain the same but their ranks change a bit. They in total observe 99.61%, an enormous fraction of overlay connections and the top 1 AS is still 3356 witnessing 64.25% of all connections. Note that the AS observing the most can see 80620 while the one observing the least can only see 13900 nodes. As for the number of connected components, to the contrary of Kad, each AS's graph

Table 4.2: Kad AS-level

Kad Day1 ISP	AS	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
Level3	3356	65.25%	100000	38192566	763.9	1	99.02%
AT&T	7018	35.33%	100000	20679083	413.6	1	99.02%
XO	2828	23.39%	100000	13691127	273.8	1	99.02%
Sprint	1239	8.32%	99983	4872140	97.5	1	99.01%
Verizon	19262	8.30%	100000	4859686	97.2	1	100.00%
Qwest	209	8.28%	100000	4848724	97.0	1	99.02%
NTT	2914	7.78%	99993	4556302	91.1	1	99.02%
BellSouth	6389	7.78%	100000	4554972	91.1	1	99.01%
AT&T	7132	6.78%	99995	3965587	79.3	1	100.00%
UUNET	701	5.38%	99937	3148400	63.0	1	88.13%
Kad Day2							
Level3	3356	66.69%	100000	39628509	792.6	1	99.02%
AT&T	7018	34.96%	100000	20772860	415.5	1	99.02%
XO	2828	24.18%	100000	14367036	287.3	1	99.02%
Qwest	209	8.35%	100000	4959076	99.2	1	99.02%
Sprint	1239	7.76%	99969	4611389	92.3	1	99.01%
BellSouth	6389	7.59%	100000	4509341	90.2	1	99.01%
Verizon	19262	7.23%	100000	4294952	85.9	1	100.00%
NTT	2914	7.06%	99988	4196433	83.9	1	99.02%
AT&T	7132	6.33%	99990	3761651	75.2	1	100.00%
UUNET	701	4.78%	99912	2839591	56.8	1	87.75%

Table 4.3: Modified Chord AS-level

ISP	AS	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
Level3	3356	64.25%	80620	112431	2.8	9639	66.22%
AT&T	7018	38.09%	54272	66650	2.5	10534	51.62%
XO	2828	22.73%	36234	39784	2.2	7470	47.03%
Verizon	19262	9.43%	17365	16494	1.9	3726	37.01%
NTT	2914	8.09%	15339	14151	1.8	3384	34.45%
Sprint	1239	7.64%	14908	13366	1.8	3602	31.16%
Qwest	209	7.20%	14642	12594	1.7	3757	27.99%
AT&T	7132	7.13%	13849	12482	1.8	2956	33.29%
BellSouth	6389	6.82%	13486	11934	1.8	3080	30.47%
UUNET	701	6.27%	13900	10978	1.6	4305	16.41%

Table 4.4: Simple ring AS-level

ISP	AS	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
Level3	3356	64.76%	79327	64755	1.6	14522	63.31%
AT&T	7018	37.51%	51316	37511	1.5	13805	46.20%
XO	2828	22.81%	32148	22805	1.4	9343	41.88%
Verizon	19262	9.30%	13632	9297	1.3	4335	36.40%
NTT	2914	8.05%	11867	8046	1.3	3821	35.60%
Sprint	1239	7.53%	11604	7532	1.3	4072	29.82%
Qwest	209	7.36%	11494	7362	1.3	4132	28.10%
AT&T	7132	7.07%	10430	7066	1.3	3364	35.49%
BellSouth	6389	6.73%	10193	6728	1.3	3465	32.01%
UUNET	701	6.17%	10831	6166	1.1	4665	13.86%

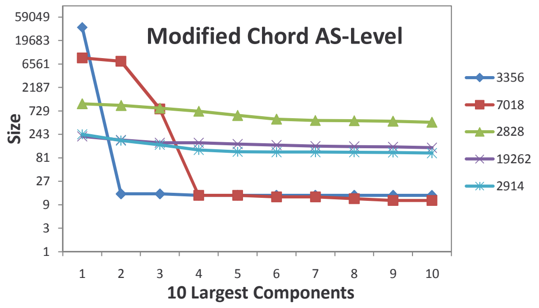


Figure 4.2: Modified Chord: 10 largest components at top 5 ASes

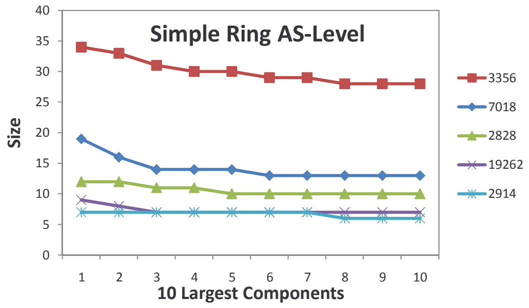


Figure 4.3: Simple ring: 10 largest components at top 5 ASes

is not well connected and has thousands of connected components. Figure 4.2 shows in log scale the sizes of 10 largest weakly-connected components at top 5 ASes. Top 1 AS 3356's largest component has 36532 nodes but all other components are very small containing 15 nodes or so. Top 2 AS 7018 has two large components with 8729 and 7506 nodes respectively and other components' sizes drop significantly. The component sizes remain stable at other ASes, all in the order of hundreds. Due to the relatively sparse structure of the modified Chord, unsurprisingly, the average degree at each AS is low—from 2.8 to 1.6, though the InO values are high—from 66.22% to 16.41%. Taking all metrics into account, AS 3356 is able to detect the P2P overlay since it can see a large portion of the overlay with typical P2P patterns, if not the entire one. If we relax the average degree threshold a bit, AS 7018 may also be a good venue to make detection efforts considering the two large connected components. We think it is hard for the rest of the ASes to do so due to their relatively fragmented views. Note that our observations on modified Chord are slightly different from those in [54] which concludes that even at the most central (top) ASes the average degrees are less than 2 and connected components are mostly of size 2 and 3 with the maximal containing 29 nodes. This difference may be attributed to the way of mapping the overlay to the underlay: they make the number of overlay nodes in each AS proportional to the size of the AS whereas we consider the geographical distribution of nodes. In addition, our AS-path inference algorithm is also different from theirs: they assume shortest paths while our inter-domain AS-pathes are derived from real-world BGP routing tables.

When it comes to the simple ring structure (Table 4.4), the top AS numbers do not change, and their ranks are the same as those for the modified Chord. 99.62% of overlay connections traverse through top 10 ASes. Though the top1 AS 3356 can see 64.76% of the total traffic, the number of nodes visible (79327) are more than the number of edges (64755), resulting in a great number of connected components

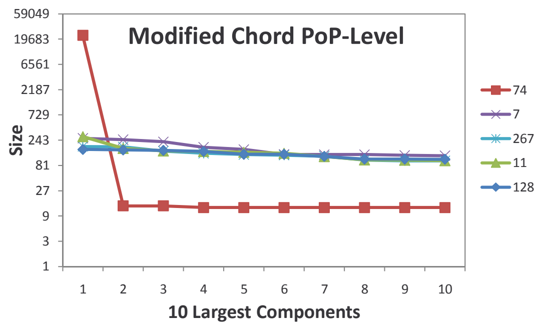


Figure 4.4: Modified Chord: 10 largest components at top 5 PoPs

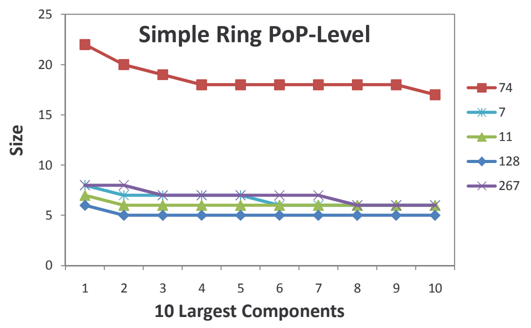


Figure 4.5: Simple ring: 10 largest components at top 5 PoPs

(14522) and small component sizes. As seen in Figure 4.3, 3356's largest component only has 34 nodes. We also verify that a majority of 3356 connected components have fewer than 10 nodes. The average degrees are all below 2, which is expected because each node only has a predecessor and a successor so that the average degree of the entire graph is only 2. Even though the InO values are high enough, detection based on scattered information at a single AS is difficult.

4.4.2 PoP-Level Analysis

At the PoP level, we also present graph features at each top PoP of three P2P structures. PoPs are represented by ID numbers and ranked by the number of unique connections going through as well. In Table 4.5, as we can see, both the top 10 PoP numbers and their ranks change slightly from day1 to day2. Top 10 PoPs account for 80.88% of overlay connections in day1 and 81.58% in day2, a slightly drop compared to that observed at top 10 ASes which can see more than 98%. This makes sense because PoPs, normally as traffic exchange points, are not able to see intra-domain traffic taking place within ASes. The top PoP 74 alone is able to observe 53.78% and 54.84% of all connections respectively in each day. Similar to the AS-level, not only almost all nodes (more than 99967) can be seen at each top PoP, but also they are weakly connected forming one single component. The average degrees and InO ratios are well above the P2P classification thresholds.

Table 4.5: Kad PoP-level

Kad Day1 PoP	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
74	53.78%	100000	31479094	629.6	1	100.00%
7	10.29%	100000	6024939	120.5	1	99.94%
435	8.27%	100000	4837622	96.8	1	98.50%
11	8.14%	99998	4763870	95.3	1	99.86%
128	7.77%	99981	4550316	91.0	1	99.52%
282	7.37%	99995	4315967	86.3	1	100.00%
4	7.27%	99977	4257513	85.2	1	99.73%
267	6.72%	99992	3934199	78.7	1	100.00%
291	6.26%	99975	3661420	73.2	1	100.00%
295	6.25%	99997	3658911	73.2	1	99.97%
Kad Day2						
74	54.84%	100000	32588327	651.8	1	100.00%
7	10.06%	100000	5976120	119.5	1	99.97%
128	8.22%	99991	4883282	97.7	1	99.66%
11	8.06%	100000	4790115	95.8	1	99.87%
291	7.49%	99997	4450255	89.0	1	100.00%
435	7.41%	100000	4404198	88.1	1	98.60%
267	7.20%	99996	4279914	85.6	1	100.00%
4	7.19%	99967	4271196	85.5	1	99.67%
282	7.07%	99992	4199285	84.0	1	99.99%
239	5.88%	99879	3491615	69.9	1	99.65%

Table 4.6: Modified Chord PoP-level

PoP	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
74	54.07%	77488	94629	2.4	16735	48.00%
7	9.27%	19927	16222	1.6	6095	21.91%
267	7.99%	14764	13981	1.9	3092	34.80%
11	7.98%	17225	13957	1.6	5334	18.75%
128	7.46%	17169	13058	1.5	5673	17.39%
4	7.25%	15962	12686	1.6	4834	20.36%
435	6.94%	13649	12151	1.8	3067	32.38%
282	6.81%	13677	11913	1.7	3184	31.41%
291	6.36%	12433	11137	1.8	2683	32.68%
295	5.84%	11877	10228	1.7	2803	29.32%

Table 4.7: Simple ring PoP-level

PoP	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
74	54.51%	75999	54506	1.4	21493	43.44%
7	9.40%	16165	9400	1.2	6765	16.30%
11	7.78%	13648	7779	1.1	5869	13.99%
128	7.63%	13765	7631	1.1	6134	10.88%
267	7.52%	11079	7521	1.4	3558	35.77%
4	7.31%	12505	7305	1.2	5200	16.83%
435	7.13%	10568	7127	1.3	3441	34.88%
282	7.08%	10587	7078	1.3	3509	33.71%
291	6.37%	9392	6373	1.4	3019	35.71%
295	5.77%	8829	5774	1.3	3055	30.80%

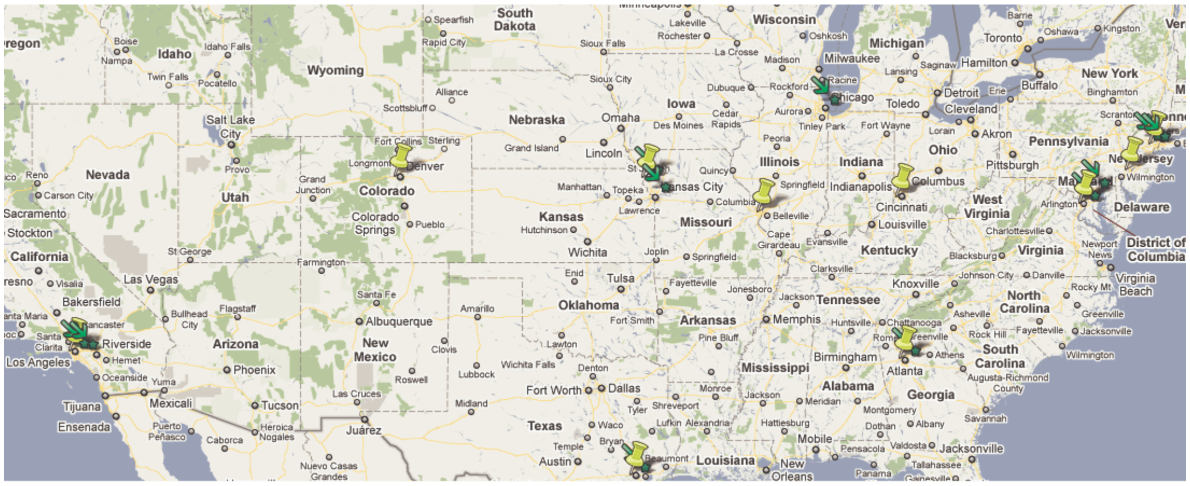


Figure 4.6: Top 10 PoPs (pins) and router rendezvous (arrows)

In the modified Chord's case as shown in Table 4.6, top PoPs are almost the same as those of Kad and only their ranks change, taking up 80.29% of overlay connections aggregately. 74 is still the top 1 PoP observing 54.07% of total connections containing 77488 nodes, but all other PoPs observe fewer than 20000 nodes. As for sizes of weakly connected components, shown in Figure 4.4 in log scale, PoP 74's largest component is of size 23153 and others are quite small. Other PoPs' component sizes are fewer than 300. Given all these statistics, if the average degree threshold can be relaxed a bit, PoP 74 can be a good place for detection.

In simple ring's case (Table 4.7), the PoP numbers are exactly the same as those of modified Chord. Figure 4.6 shows the geographical locations of the top 10 PoPs represented by pin icons. Note that they hardly change across the three structures and their locations are distributed throughout the US. 89.25% of overlay connections reach top 10 PoPs with 54.51% traversing PoP 74. Despite the fact that half of overlay connections can be observed at PoP 74, similar to the AS-Level, the number of edges is smaller than the number of nodes. The largest component of PoP 74 is very small containing 22 nodes (Figure 4.5). It is the same case for all other top PoPs. Though InO values are moderate, low average degrees and a good many small connected components can prevent the P2P structure from being captured at any PoP.

4.4.3 Router-Rendezvous-Level Analysis

At the router-rendezvous level, we present results the same way as before. Router rendezvous are denoted by ID numbers and ordered by the number of unique overlay connections observed. For the Kad structure, as shown in Table 4.8, the top 10 router rendezvous are the same throughout the two days, altogether, see 89.75% of total connections in day1 and 89.27% in day2. The top 1 router rendezvous number 2 is reached by 68.77% of all connections in day1 and 68.91% in day2. A majority of nodes (more than 98579) appear in the graph as one giant component at each top router rendezvous. In addition, high average degrees and InO values make detection feasible.

Table 4.8: Kad router-rendezvous-level

Kad Day1 Router	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
2	68.77%	100000	40251799	805.0	1	100.00%
2164	14.91%	99959	8728267	174.6	1	98.96%
12	11.90%	99997	6967203	139.3	1	84.22%
98	11.75%	100000	6874621	137.5	1	100.00%
222	9.26%	100000	5419174	108.4	1	99.99%
8919	8.30%	100000	4855632	97.1	1	98.50%
745	7.82%	99997	4579803	91.6	1	99.85%
82	7.33%	99978	4288889	85.8	1	99.74%
47	6.99%	98858	4090556	82.8	1	92.32%
88	6.67%	99997	3904395	78.1	1	99.71%
Kad Day2						
2	68.91%	100000	40945772	818.9	1	100.00%
2164	14.52%	99959	8626011	172.6	1	99.32%
12	11.57%	99989	6876147	137.5	1	83.77%
98	11.28%	100000	6702210	134.0	1	100.00%
222	9.05%	100000	5379049	107.6	1	99.98%
8919	7.41%	100000	4404198	88.1	1	98.60%
745	7.67%	100000	4559186	91.2	1	99.86%
82	7.24%	99973	4304038	86.1	1	99.68%
88	6.53%	99996	3881327	77.6	1	99.61%
47	6.18%	98579	3671730	74.5	1	90.15%

Let us take a look at the modified Chord at the router-rendezvous level (Table 4.9). There is one new router rendezvous in the top 10 list that does not appear in that of Kad's and the ranks of the two lists are quite similar. Top 10 router rendezvous carry 89.96% of total connections and the top 1 router rendezvous is still 2 accounting for 68.76% of connections including 88913 nodes. As for the sizes of weakly connected components, the trend does not differ much from that at the AS- or PoP-level. The top 1 router rendezvous's largest connected component is of a big size—35943 nodes (Figure 4.7 in log scale) and other components have small sizes (fewer than 15). With a distinctive average degree and high InO value, this router rendezvous is a reasonable venue for capturing the modified Chord.

Table 4.9: Modified Chord router-rendezvous-level

Router	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
2	68.76%	88913	120337	2.7	13816	59.33%
2164	15.00%	29299	26245	1.8	8964	25.60%
12	11.57%	23682	20247	1.7	7629	20.07%
98	11.33%	21641	19821	1.8	5586	31.07%
222	8.73%	17779	15280	1.7	4771	27.68%
745	7.59%	16673	13275	1.6	5286	17.14%
82	7.29%	16133	12758	1.6	4926	19.98%
8919	6.94%	13649	12151	1.8	3067	32.38%
88	6.26%	12913	10962	1.7	3364	25.96%
57	6.16%	13606	10784	1.6	4029	19.42%

Table 4.10: Simple ring router-rendezvous-level

Router	Percentage	# of Nodes	# of Edges	Avg Degree	# of Conn Comp	InO
2	68.89%	88161	68885	1.6	19276	56.27%
2164	15.12%	25513	15122	1.2	10391	18.54%
12	11.35%	20126	11351	1.1	8775	12.80%
98	11.28%	17720	11275	1.3	6445	27.26%
222	8.93%	14243	8933	1.3	5310	25.44%
745	7.42%	13218	7419	1.1	5799	12.26%
82	7.36%	12653	7356	1.2	5297	16.27%
8919	7.13%	10568	7127	1.3	3441	34.88%
88	6.10%	9762	6102	1.3	3660	25.02%
47	6.06%	9669	6061	1.3	3608	25.37%

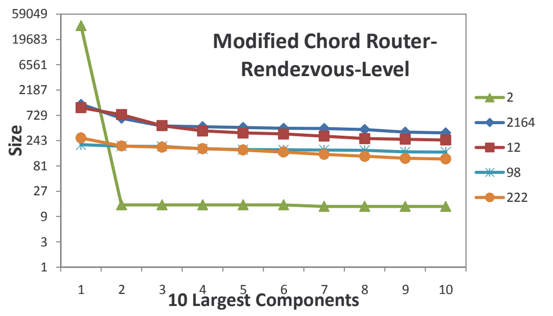


Figure 4.7: Modified Chord: 10 largest components at top 5 locations

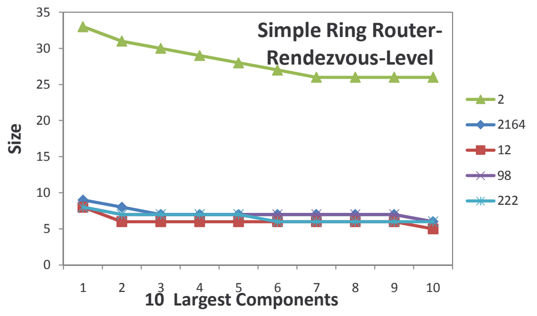


Figure 4.8: Simple ring: 10 largest components at top 5 locations

Finally, for the simple ring structure (Table 4.10), the top router rendezvous list is the same as that of Kad. Figure 4.6 illustrates all top router rendezvous for the three structures, each represented by an arrow with a star. Note that some of them are co-located with the top PoPs: in fact, PoPs are a subset of router rendezvous. Top 10 router rendezvous observe 80.54% of all connections and router 2 sees 68.89% of them. With more nodes than edges at each top router rendezvous, it is difficult to get a good view of the P2P overlay. Similar to AS- and PoP-level, the top 1 router rendezvous's largest component contains 33 nodes. The average degrees are unsurprisingly low, insufficient for detection.

4.4.4 Insights from Analysis

From the above analysis, we have several key observations worth noting. First, the visibility of Kad's overlay and structure at the top places of all three levels is good enough for detection; the modified Chord's P2P characteristics can be captured by a few top locations but not all; and the information of the hypothetical simple ring's topology at all levels is quite fragmented and hardly useful for detection. From the attacker's viewpoint, in terms of efficiency, Kad has the most efficient routing: contacting $O(\log N)$ nodes during a search (where N is the size of the network); the modified Chord can achieve $O(\log^2 N)$ hops; and the simple ring is the worst, requiring $O(N)$ steps. From resilience's perspective, the Kad network is shown to be robust to a few types of mitigation strategies such as cutting off random nodes and removing peers learnt from bots' peer lists [36]; the simple ring structure is evidently fragile—removing a couple of nodes can disconnect the overlay; and the modified Chord structure hits the middle ground: not as resilient as Kad but better than the simple ring. We believe that, while constructing a P2P botnet, the attacker needs to strike a balance between resilience or efficiency and the ability to evade detection. Although the simple ring can hide its traffic well at various network components, to build upon this structure the botnet has to compromise resilience and C&C efficiency. The modified Chord makes a tradeoff though its structural properties cannot be concealed at some locations. Kad was successfully utilized by the Storm botnet, but given our detection strategy, to use it for a future botnet, the attacker has to come up with techniques to mask its P2P patterns.

Second, from detection's perspective, AS-level provides better overlay views than PoP- and router-rendezvous-level do, but is less practical than the other two for actual detection deployment. Since AS is only a logical concept, capturing all connections within one single AS requires collaboration and synchronization among multiple physical devices at different geographical locations, which renders it highly impractical. From our analysis, we can see that at the PoP-level, detecting Kad and the modified Chord is very likely though the latter is only visible to the top 1 PoP. Compared to ASes and router rendezvous, PoPs observe less traffic due to the invisibility of traffic within ASes (intra-domain traffic). Moreover, the number of PoPs is small so that the points available for monitoring are limited. Among the three, router rendezvous make a tradeoff. Their detection capabilities are comparable to PoPs' and they can observe intra-domain traffic with more monitoring points available, making detection more feasible.

4.5 Discussions

Thus far, we have measured the effectiveness of identifying P2P overlay traffic at various network components. For actual implementation of the detection at the Internet infrastructure, several challenges remain to be addressed.

First, since P2P networks implementing the same protocol may not be distinguishable at the structure-level via graph analysis, our techniques will also identify regular P2P file-sharing topologies. To avoid misclassifying such regular P2P networks as botnets, we can perform preprocessing including flow filtering and clustering [52] based on known patterns of regular P2P networks such as port numbers. Also, bots identified locally in edge networks are helpful as their presence in a communication graph makes other nodes suspicious as well, so our approach may need assistance from detection mechanisms at the edge to further confirm that a graph is indeed formed by a botnet. However, if the botnet is immersed into an existing regular P2P network, detecting it solely by graph analysis at the infrastructure level would be challenging and other information is thus needed for effective detection.

Second, in the presence of a huge traffic volume, some connections could not be captured due to sampling. For densely-connected topologies such as Kad, it may not be a problem. But for the modified Chord and simple ring's cases, it will complicate the detection. We plan to dig deeper into this issue in the future.

Third, after identifying nodes of a botnet, to further mitigate or contain bots, we need to come up with efficient and effective techniques that can accommodate a large volume of traffic at the infrastructure level with minimal impact on the legitimate traffic. In the edge or local networks, fine-grained information of a particular node is available to detection mechanisms, and all incoming and outgoing traffic of the node can be controlled. Thus, after detection, taking the suspected node offline is not a difficult task. However, at the infrastructure level, a single network component may not have the ability and the confidence to remove a node completely so that advanced response mechanisms other than simply filtering or blacklisting are needed.

Finally, our models regarding the Internet infrastructure are abstracted from real-world datasets, so the accuracy depends on how well the datasets characterize the behavior and the state of the Internet, which could be error-prone. Moreover, some datasets may be outdated and may not reflect the current state of the Internet due to its fast-evolving nature. Therefore, these factors have to be taken into account when the infrastructure-level detection is put in practice.

4.6 Conclusion

As P2P structures become a popular choice for recent botnets, especially large-scale ones, detection mechanisms have to keep up with this change and identify bots in an efficient and effective manner. In this paper, we propose detection of P2P botnets at a high-level—the infrastructure-level by analyzing their structural properties from a graph perspective. We construct three different P2P overlay topologies: Kad, the modified Chord and the simple ring. These overlays are mapped to the AS-level underlays and their respective AS-, PoP- and router-rendezvous-paths are inferred. Finally, we inspect these network components individually to measure their capability in identifying the P2P botnets. We find that detection at any of the three network components has its advantages and drawbacks. Overall, router-rendezvous-level detection is able to strike a balance between detection capability and feasibility. Also, a botnet needs to make a tradeoff between resilience and stealthiness.

CHAPTER V

The Next-Generation Botnet

5.1 Introduction

Botnets have become a most serious security threat to the Internet and the personal computer (PC) world. Although they have not yet caused major outbreaks in the wild, attacks on cellular networks and devices have recently grown in number and sophistication. With the rapidly-growing popularity of smartphones, such as the iPhone and Android-based phones, there has been a drastic increase in downloading and sharing of third-party applications and user-generated content, making smartphones vulnerable to various types of malware. Smartphone-based banking services have also become popular without protection features comparable to those on PCs, enticing cyber crimes. There are already a number of reports on malicious applications in the Android Market [7]. Although the Android platform requires that applications should be certified before their installation, its control policy is rather loose—allowing developers to sign their own applications—so that attackers can easily get their malware into the Android Market. The iPhone's application store controls its content more tightly, but it fails to contain jailbroken iPhones which can install any application and even run processes in the background. As smartphones are increasingly used to handle more private information with more computing power and capabilities but without adequate security and privacy protection, attacks targeting mobile devices are becoming more sophisticated. Since the appearance of the first, proof-of-concept mobile worm, Cabir, in 2004, we have witnessed a significant evolution of mobile malware. The early malware performed tasks, such as infecting files, replacing system applications and sending out SMS or MMS messages. One malicious program is usually capable of only one or two functions. Although the number of mobile malware families and their variants has been growing steadily in recent years, their functionalities have remained simple until recently.

SymbOS.Exy.A trojan [85] was discovered in February 2009 and its variant SymbOS.Exy.C resurfaced in July 2009. This mobile worm, which is said to have "botnet-esque" behavior patterns, differs from other mobile malware because after infection, it connects back to a malicious HTTP server and reports information of the device and its user. The Ikee.B worm [51] that appeared late November 2009 targets jail-broken iPhones, and has behavior similar to SymbOS.Exy. Ikee.B also connects to a control server via HTTP, downloads additional components and sends back the user's information. With this remote connection, it is possible for attackers to periodically issue commands to and coordinate the infected devices to launch large-scale attacks. In March 2011, over 50 applications found to contain a type of malware called "DroidDream" were removed from the Android Market [7]. This malware was able to root the infected device and steal sensitive information. It was speculated that the end goal of DroidDream was to create a botnet [11]. Observing the trend of recent mobile malware, we expect that mobile botnets will likely become a serious threat to smartphone soon.

Similar to PC-based botnets, mobile botnets also require three key components: vectors to spread the bot code to smartphones; a channel to issue commands; and a topology to organize the botnet. Compared to their PC counterparts, mobile devices have the following unique features that botnets can take advantage of: (1) they communicate via multiple vectors including SMS/MMS messages, Bluetooth, aside from the conventional IP network. (2) they move around frequently, and it is generally difficult to find vantage points that can observe devices' all activities. (3) they have limited security protection features both in the host and in the cellular network. These features together present a good opportunity for next-generation botnets to exploit. So in this chapter, we focus on the design of a proof-of-concept botnet that makes the most of mobile services and is resilient to disruption. Within this mobile botnet, all C&C communications are done via SMS messages since SMS can reach almost every mobile phone anywhere anytime with little scrutiny and restriction from mobile carriers. To hide the identity of the botmaster, there are no central servers dedicated to command dissemination that is easy to be identified and then removed. Instead, we adopt a P2P topology that allows botmasters and bots to publish and search for commands in a P2P fashion, making their detection and disruption much harder. We will also briefly discuss a few defensive strategies and how the detection solutions proposed in previous chapters can be applied.

5.1.1 Contributions

Our contributions are three-fold. First, to the best of our knowledge, we are the first to design mobile botnets with focuses on both C&C protocol and topology by integrating the SMS service and the P2P topology. The main intent of this work is to shed light on potential botnet threats targeting smartphones. Since current techniques against PC botnets may not be applied directly to mobile botnets, our proposed mobile botnet design makes it possible for security researchers to investigate and develop countermeasures before mobile botnets become a major threat. Second, we present a method to carefully disguise C&C content in spam-looking SMS messages. Using this approach, the botnet can stealthily transmit C&C messages without being noticed by phone users. Third, we test and compare two P2P architectures that can be used to construct the topology of our mobile botnet on an overlay simulation framework, and finally propose the architecture that best suits mobile botnets.

5.1.2 Organization

The remainder of the chapter is organized as follows. Section 5.2 describes related work in this domain. Section 5.3 details the proof-of-concept design of our mobile botnet. Section 5.4 presents our simulation and evaluation results. Section 5.5 discusses potential countermeasures against the mobile botnets. The chapter concludes with Section 5.6.

5.2 Related Work

The research areas most relevant to our work are P2P-based botnets and botnet C&C evaluation. Wang *et al.* [89] proposed the design of an advanced hybrid P2P botnet that implemented both push and pull C&C mechanisms and studied its resilience. In [90] they conducted a systematic study on P2P botnets including bot candidate selection and network construction, and focused on index poisoning and Sybil attacks. Overbot [81] is a botnet protocol based on Kademlia. The strength of this protocol lies in its stealth in the communication between the bots and the botmaster leveraging a public-key model. Davis *et al.* [36] compared the performance of Overnet with that of Gnutella and other complex network models under three disinfection strategies. Singh *et al.* [78] evaluated the viability of email communication for botnet C&C. Nappa *et al.* [66] proposed a botnet model exploiting Skype's overlay network to make botnet traffic undistinguishable with legitimate Skype traffic. All of these dealt with botnets in the PC world, while our work targets mobile botnets, in which C&C channel and network structure requirements are different, in view of unique services and resource constraints on smartphones. Dagon *et al.* [35] proposed key metrics to measure botnets' utility for conducting malicious activities and considered the ability of different response techniques to disrupt botnets.

There are numerous efforts on mobile malware focusing on vulnerability analysis and attack measurements. The former investigates ways of exploiting vulnerable mobile services, such as Bluetooth and MMS [30, 72], while the latter characterizes the feasibility and impact of large-scale attacks targeting mobile networks, mostly Denial of Service (DoS) attacks [38]. There are a few recent papers treating the idea of mobile botnets. In [86], the focus is on the attack aspect—whether compromised mobile phones can generate sufficient traffic to launch a DoS attack. Singh *et al.* [77] investigated using Bluetooth as a C&C to construct mobile botnets without any analysis on their network structure. Hua *et al.* [50] proposed a SMS-based mobile botnet using a flooding algorithm to propagate commands with the help of an internet server. The use of the server may lead to single-point-of-failure, meaning that whenever the server is identified and removed, the botnet is prone to disruption. Mulliner *et al.* [64] demonstrated the ways to command and control botnets via SMS or IP-based P2P networks using a tree topology. Under such topology, when a node fails, all of its subnodes will be isolated from the botnet, difficult to get commands. Weidman [92] also considered utilizing SMS messages for botnet C&C and presented a method to conceal malicious SMS messages from users on smartphones. It is worth noting that, different from all these works, our SMS-based botnet is built upon a decentralized P2P topology, without assistance from any central servers. The integration of SMS and P2P makes our botnet stealthy and resilient to disruption.

5.3 Mobile Botnet Design

We now present the detailed design of a proof-of-concept mobile botnet. The botnet design requires three main components: (1) vectors to spread the bot code to smartphones; (2) a channel to issue commands; (3) a topology to organize the botnet. We will briefly overview approaches that can be used to propagate malicious code and then focus on C&C and topology construction.

5.3.1 Propagation

The main approaches used to propagate malicious code to smartphones are user-involved propagation and vulnerability exploits.

In the first approach, the most popular vector is social engineering. Like their PC counterparts, current smartphones have frequent access to the Internet, becoming targets of malicious attacks. Thus, spam emails and MMS messages with malicious content attachments, or with embedded links pointing to websites hosting the malicious code, can easily find their way into a mobile phone's inbox. Without enough caution or warning, a mobile phone user is likely to execute the attachments or click those links to download malicious programs. The advantage of such schemes is that they can reach a large number of phones. Nevertheless, as smartphones run on a variety of operating systems, we expect multiple versions of bot code prepared to guarantee its execution. Another user-involved propagation vector can be Bluetooth, which utilizes mobility. Mobile phone users move around so that the compromised phones can use Bluetooth to search for devices nearby and after pairing with them successfully, try to send them malicious files.

Exploiting vulnerabilities to spread malicious code is common in the PC world. However, since there are various mobile platforms and most of them are closed-source, it is difficult to find vulnerabilities in real deployments. To date, some vulnerabilities have been discovered. For example, the HTC's Bluetooth vulnerability, which allows an attacker to gain access to all files on a phone by connecting to it via Bluetooth, was disclosed by a Spanish security researcher [9]. Mulliner *et al.* [63] discovered a way of directly manipulating SMS messages on different mobile platforms, without necessarily going through the mobile provider's network. In both cases, OS vendors immediately released patches to the public after the vulnerabilities were publicized, leaving few opportunities for a real exploit in the wild. Once launched in their targets, vulnerability exploits always have a higher success rate than that of user-involved approaches. As mobile platforms open up and mobile applications and services become abundant, vulnerability exploits will play a major role in mobile malware propagation.

5.3.2 Command and Control

5.3.2.1 Why SMS

In our mobile botnet, SMS is utilized as the C&C channel, i.e., compromised mobile bots communicate with botmasters and among themselves via SMS messages. Botnets in the PC world mostly rely on IP-based C&C delivery. For example, traditional botnets use centralized IRC or HTTP protocol, whereas newly-emerged botnets take advantage of P2P communication. Unlike their PC counterparts, smartphones can hardly establish and maintain steady IP-based connections with one another. One reason is that they move around frequently. Another reason is that private IPs are normally used when smartphones access networks, especially EDGE and 3G networks, meaning that accepting incoming connections directly from other smartphones is a difficult task. Given this limitation, if a mobile botnet considers an IP-based channel as C&C, it needs to resort to centralized approaches in which bots connect to central servers to obtain commands. Such approaches, however, are vulnerable to disruption because the servers are easy to be identified by defenders. Thus, to construct a mobile botnet in a more resilient manner, a non-IP-based C&C is needed.

There are a few advantages for choosing SMS as a C&C channel. First, SMS is ubiquitous. It is reported that SMS text messaging is the most widely used data application on the planet, with 2.4 billion active users, or 74% of all mobile phone subscribers sending and receiving text messages on their phones [79]. When a mobile phone is turned on, this application always remains active. Second, SMS can accommodate offline bots easily. For example, if a phone is turned off or has poor signal reception in certain areas, its SMS communication messages will be stored in a service center and delivered once the phone is turned back on or the signal becomes available. Third, malicious content in the C&C communication can be hidden in SMS messages. According to a survey in China [5], 88% of the phone users polled reported they had been plagued by SMS spamming. As SMS spamming becomes prevalent, bots can encode commands into spam-looking messages so that users will not suspect. Last but not least, currently there are multiple ways to send and receive free SMS messages directly on smartphones [23, 24] or through some web interfaces. We will describe such methods in Section 5.3.2.4. Even when the free texting is unavailable, as many phone users use SMS plans to avoid per-message charge and in some countries incoming messages are free of charge, with the design goal of minimizing the number of SMS messages we expect that using SMS as C&C will not incur considerable costs.

5.3.2.2 SMS Overview

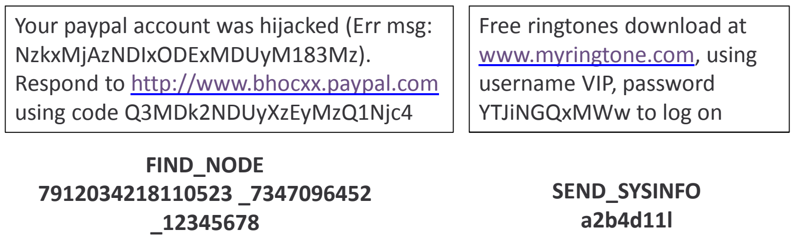
Before discussing how to use SMS for C&C, we briefly describe the implementation of SMS. When a user sends a SMS message, the mobile phone sends it along with the address of the Short Message Service Center (SMSC) over the air interface to a Base Station Subsystem (BSS) of the service provider. The BSS then sends the text to the Interworking Mobile Switching Center (MSC) of the SMSC. The Interworking MSC returns an acknowledgment indicating success or failure, and passes the message to the actual Service Center (SC) of the SMSC for its storage and/or delivery. When it delivers the message, the SMSC queries a database called *Home Location Register* (HLR) to determine the location of the mobile phone. If the phone is available, the message is forwarded through a few steps to the MSC which finally delivers the text message over the air interface through its BSS to the recipient's phone.

5.3.2.3 Protocol Design

Our goal is to let a phone that has installed our bot code perform activities according to the commands in SMS messages without being noticed by the user, if possible. In our design, every compromised phone has an 8-byte passcode. Only by including this passcode into the SMS messages, can other phones successfully deliver C&C information to this particular phone. Upon receipt of a SMS message, this phone searches for its passcode and pre-defined commands embedded in the message to tell if it is a C&C message. If found, the commands are immediately executed by the phone. Two issues need to be addressed here. First, how are passcodes allocated among compromised phones? Second, how to make C&C SMS messages appear harmless so that users may not notice the malicious content?

In our botnet, passcodes are allocated by botmasters to segment a botnet into sub-botnets, each with a different function. For example, one sub-botnet is responsible for sending out spam messages, while another is in charge of stealing personal data and transferring them to a malicious server. Each sub-botnet will be identified by its unique passcode that is hard-coded into the bot's binary. In other words, all bots within the same sub-botnet share the same passcode so that they can communicate with one another and also with the botmaster. Using a unique passcode for each bot will be more secure than using one passcode for an entire sub-botnet because in the latter case, the passcode will be discovered more easily. However, there is a tradeoff: using a unique passcode will add more overhead due to the pairwise passcode exchange before each communication. The additional cost is undesirable since our goal is to minimize the number of SMS messages to be sent.

Not only do we require a passcode included in each SMS communication message, but also we encode commands to make it difficult for a user to figure them out. In fact, on the Android platform, it is possible for an application to send out SMS messages stealthily, to get immediate notification of every incoming SMS message by registering



Your paypal account was hijacked (Err msg: NzkxMjAzNDIxODExMDUyM183Mz). Respond to http://www.bhocxx.paypal.com using code Q3MDk2NDUyXzEyMzQ1Njc4

FIND_NODE
7912034218110523 _7347096452
_12345678

Free ringtones download at www.myringtone.com, using username VIP, password YTJiNGQxMWw to log on

SEND_SYSINFO
a2b4d11l

Figure 5.1: Disguised SMS messages

itself as a background service and to read and execute commands or even delete the message before the user sees it. We still want to hide the C&C messages because other mobile platforms are more restricted than Android; they may not allow our bots to both send and receive SMS messages without notifying the user. If malicious messages show contents directly, they will be easily captured and manipulated by defenders. To evade such detection, we want to make a command-embedded SMS message look like a common message such as a spam message. There are benefits of using spam-like messages to transmit C&C. As pointed out in [8], cellular carriers cannot simply block offending SMS messages because the senders have paid for the messages and the carriers fear permanent deletion of legitimate messages when there are no spam folders available. We will present a real-world experiment in Section 5.4.3. Even if in the future the carriers filter out spam messages and dump them into spam folders, similar to the email filtering, spam messages can still reach the target phones by going to the spam directory, which actually helps hide the C&C because users tend to ignore spam.

Considering the fact that each SMS message only contains up to 160 characters, commands in our botnet are concise. For example, "FIND_NODE" instructs a bot to return the phone numbers of certain nodes; "SEND_SYSINFO" asks a bot to reply with system information. To disguise messages, each command is mapped to one spam template. Additional information such as the phone number and the aforementioned passcode are variables in the templates, and they are Base64-encoded.

Figure 5.1 shows two disguised SMS messages. The first one is a "FIND_NODE" message (146 characters) with passcode 12345678 requiring the recipient to locate a bot whose ID is 7912034218110523, and the result should be returned to the bot whose phone number is (734)7096452. NzkxMjAzNDIxODExMDUyM183Mz and Q3MDk2NDUyXzEyMzQ1Njc4—two random strings together—are the Base64-encrypted 7912034218110523_7347096452_12345678. The entire encoded string is split into two—disguising one as an error message and the other as a code—making it resemble a spam message. The second example is a "SEND_SYSINFO" message (98 characters) with a passcode a2b4d11l. This template is different from that of "FIND_NODE" message. The passcode is also Base64-encoded and appears as a password in the disguised message. To decode messages, each bot keeps a command-template mapping list. Since only tens of commands are needed in our botnet, this list is not long. To make detection harder, one command message can correspond to different spam templates and the templates can be updated periodically. As just shown, a command along with additional information can be easily embedded into one SMS message which appears to be a spam, familiar to today's phone users, so users are likely to ignore such messages even if they open and read them. If users choose to delete these messages, it will not cause any problem to the botnet because the commands have already been executed upon their receipt. Without monitoring phone behavior or reverse engineering, defenders may have difficulty in figuring out the mapping between spam templates and commands.

5.3.2.4 Sending SMS Through the Internet

Although sending SMS messages through the cellular networks is always possible, the botmasters want to hide their identity and lower costs as much as possible. To achieve this goal, botmasters can use the Internet to disseminate C&C messages to the mobile botnet. There are several ways to do this. Many advertisement-based websites provide free SMS services. Botmasters can type in messages via these websites and have them sent to mobile bots, feasible for low-volume messaging. Using such services does not require the sender's mobile number, an email address is sufficient if a reply is expected. If the botnet is large, botmasters need to create an account with mobile operators or SMS service providers to make high-volume messaging possible at the lowest price. Usually, this can be done by sending and/or receiving SMS messages via email through a SMS gateway connecting directly to a Mobile Operator's SMSC (Short Message Service Center). Currently, smartphone applications such as [24, ?] offer free domestic and international text messaging when the phone is connected to a WiFi and support both one-on-one and group texting. The user only needs to provide a screen name to send and receive messages without revealing its identity. Both the botmasters and bots can take advantage of such a service whenever possible to avoid messaging costs.

To sum up, using SMS messages as the C&C is a viable solution for a mobile botnet. Not only is SMS ubiquitous to every mobile phone, but botmasters and bots are also able to disguise SMS messages, send bulk messages from the Internet at very low cost while hiding their identities. Thus, using SMS is both economical and efficient for the botnet.

5.3.3 Mobile Botnet Topology

In the previous section we have described the way SMS messages form the C&C communication in our mobile botnet. In what follows, we introduce P2P topologies that may be utilized to organize the botmaster and bots for publishing and retrieving commands, and describe how to leverage existing P2P architectures to meet the need for mobile botnet construction.

5.3.3.1 Possible Topologies

Similar to botnets in the PC world, a mobile botnet can be structured in a traditional centralized way or in a newly-emerged decentralized P2P fashion. In the first approach, botmasters hard-code into each bot's executable a set of phone numbers that are under their direct control. When a mobile phone is converted to a bot, it contacts those hard-coded phones to request commands or wait for commands to be pushed to them. Such a centralized topology is easy to implement but not resilient to disruption. Obviously, once defenders obtain these phone numbers, they can track down the botmasters and then disable the botnet; making the botnet susceptible to a single-point-of-failure. To make our botnet robust to defenses, we adopt a P2P structure instead.

Currently, there are several structures for P2P networks; they can be divided into three categories: centralized, decentralized but structured, and unstructured. Centralized P2P networks have a constantly-updated directory hosted at central locations. Peers query the central directory to get the addresses of peers having the desired content. This structure is similar to the traditional centralized botnet architecture and hence vulnerable to the central-point-of failure. Decentralized but structured P2P networks have no central directory and contents are not placed at random nodes but at specific locations. The most common systems in this category are Distributed-Hash-Table (DHT)-based P2P networks, ensuring that any peer can efficiently route a search to some peer with the desired content. One notable implementation is Kademlia [61], used by several current P2P applications, such as eMule and BitTorrent. Decentralized and unstructured P2P networks have neither central directories nor control over content placement. If a peer wants to find certain content in the network in old protocols such as Gnutella, it has to flood its query to the entire network to find peers sharing the data. To address the scalability issues, current unstructured networks adopt different query strategies to avoid flooding. There have also been extensive studies on how to make Gnutella-like systems scalable. One such design is Gia [32].

5.3.3.2 Design

Both structured and unstructured P2P architectures can be modified to suit our need for the mobile botnet because their decentralized nature hides the botmaster's identity. Since the mobile botnet design should consider not only robustness but also feasibility and efficiency on smartphones, we need to compare these two architectures to see which is more suitable. Specifically, we base our structured and unstructured botnet topology on Kademlia and Gia, respectively, for comparison. Note that in our botnet, bots obtain commands mainly in a *pull* style, i.e., the botmaster publishes commands and bots are designed to actively search for these commands. The other possible mechanism for command transfer is *push*, meaning that bots passively wait for commands. We prefer *pull* to *push* because *push* will get malicious activities exposed easily. That is, under *push* many SMS messages are sent out from one or a few central nodes, whereas *pull* can be implemented in a more distributed fashion. In what follows, we overview each protocol and describe our design.

Kademlia is DHT-based and has a structured overlay topology, in which nodes are identified by node IDs generated randomly and data items are identified by keys generated from a hash function. Node IDs and keys are of the same length (128-bit). Data items are stored in nodes whose IDs are close to data items' keys. The distance between two identifiers, x and y, is calculated by bitwise exclusive or (XOR) operation: $d(x, y) = x \oplus y$. For each $0 \leq i < 128$, each node keeps a list for nodes of distance between 2^i and 2^{i+1} from itself. This list is called a k-bucket, and can store up to k elements. There are four types of RPC messages in Kademlia: PING, STORE, FIND_NODE and FIND_VALUE. PING checks whether a node is online. STORE asks a node to store data. FIND_NODE provides an ID as an argument and requests the recipient to return k nodes closest to the ID. FIND_VALUE behaves similarly to FIND_NODE. The only exception is that when a node has the data item associated with the key, it returns the data item. Since there is no central sever, each node has a hard-coded peer list in order to bootstrap into the network.

Considering the differences between smartphones and personal computers as well as the SMS C&C channel we adopt, we modify Kademlia's design to be suitable for our mobile botnet's structured overlay construction. First, we do not use PING messages to query whether a node is alive and should be removed from its k-bucket. One reason for this is that SMS messages transmitting C&C can always reach their recipients even if these phones are not online (messages are stored in the SMSC for later delivery). The other reason is that our design tries to minimize the number of messages sent and received. Removing PING messages effectively reduces C&C traffic and thus, the possibility of being noticed by phone users and defenders. Second, instead of being randomly generated, a node ID is constructed by hashing its phone number, similar to the notion in Chord [82] that a node ID is the hash of its IP address. Doing so can undermine the effectiveness of Sybil attacks in which defenders add nodes to join the botnet to disrupt C&C transmission. Evidently, if node IDs are allowed to be randomly chosen, defenders will take advantage of this by selecting IDs close to command-related keys to ensure a high probability that these sybil nodes are on the route of command search and publish queries. In addition, the absence of an authentication mechanism in Kademlia, meaning that anyone can insert values under specific keys, presents an opportunity for defenders to launch index poisoning attacks by publishing fake values under command-related keys once they know these keys, in order to disrupt C&C. We thus use a public key algorithm to secure the command content. While publishing a command, the publisher (the botmaster) needs to attach a digital signature to that command. The signature is the hash value of the command signed by the botmaster's private key. Its corresponding public key is hard-coded in each bot's binary. In this way, bots that will store the command are able to verify that the command is indeed from the botmaster not anyone else.

Gia improves Gnutella protocol and has an unstructured overlay topology. Since Gnutella has a scaling problem due to the flooding search algorithm, Gia modifies Gnutella's design and improves its scalability significantly. There are four key components in Gia's design: (1) a topology adaptation protocol to put most nodes within short reach of high-capacity (able to handle more queries) nodes by searching and adding high-capacity and high-degree nodes as neighbors; (2) an active flow control scheme to avoid overloaded nodes by assigning flow-control tokens to nodes based on capacity; (3) one-hop replication to maintain pointers to the content offered by immediate neighbors; (4) a search algorithm based on biased random walks directing queries to nodes that are likely to answer the queries.

Our design of unstructured overlay topology is based on Gia as mentioned before. Our design removes the one-hop replication scheme because it requires each node to index the content of its neighbors and to exchange this information periodically. This scheme may help reduce the number of hops for locating a command, but will incur additional storage and computation overheads. Moreover, each SMS message has a limited length so that the exchange of index information cannot be done with a single message but requires multiple messages, increasing the number of messages generated. In our mobile botnet, the drawbacks of using such a scheme will outweigh its benefits, and we thus opt out of this scheme. Three other components are important to our botnet because their combination ensures queries to be directed to high-capacity nodes that can provide useful responses without getting overloaded. This is desirable, especially in a mobile phone network, since smartphones also have different capacities under different situations. For example, in a poor-signal area or when the phone is on a voice call (SMS messages use the same control channel as voice calls for delivery), the phone's capability of handling SMS messages is lowered, so it can only answer fewer queries. Overloading mobile bots is also a concern. If one bot receives/sends a large number of SMS messages during a short period of time, its battery can be drained quickly, and draw the user's attention. Overloading can be prevented using the flow-control scheme in Gia. Another design choice worth mentioning is that similar to the modified Kademila, a digital signature is attached to every command to be published.

5.4 Evaluation

5.4.1 Comparing Two P2P Structures

We now describe our simulation study of structured and unstructured P2P architectures for mobile botnets and compare their performances. In the simulation, all nodes are assumed to have already been infected by the vectors described in Section 5.3.1. Our evaluation focus is not on how the malicious bot code propagates, but on how the botnet performs under two different P2P structures.

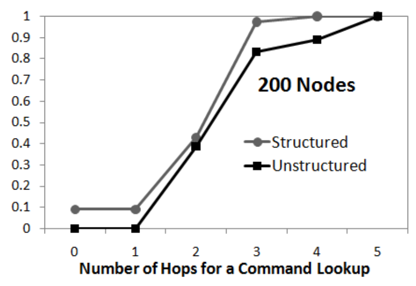
We modified OverSim [69], an open-source overlay network simulation framework, to simulate mobile botnets with the two P2P structures. While comparing P2P structures' performances, logical connections (SMS activities) among mobile nodes matter most, i.e., what we care is the overlay network not the underlying physical network. For example, the fact that mobile bots move around is not important in our simulation because the change of geographic location hardly affects bots' SMS message sending/receiving.

The metrics we use to measure performance are: number of overlay hops needed for a command lookup; total number of SMS messages sent (number of those sent = number of those received) when a botmaster-issued command is acquired by every node; percentage of total number of SMS messages sent by each node during this entire command-lookup; and message delay (from the start of the query until a command is received). These metrics reflect how well each architecture meets the requirement of our mobile botnet, namely, minimizing the number of SMS messages sent and received, load-balancing and locating commands in a timely manner.

The churn (participant turnover) model we adopted in the simulation is the lifetime churn. In this model, on creation of a node, its lifetime will be drawn randomly from a Weibull distribution which is widely used to characterize a node's lifetime. When the lifetime is reached, the node is removed from the network. A new node will be created after a dead time drawn from the same probability distribution function. We set the mean lifetime to 8*3600=28800s, assuming that each phone will stay connected to the botnet for an average of 8 hours. Considering the unavailability of real field data on mobile phones' online behavior, we made this rough estimate. We will later evaluate the effect of different mean lifetimes on the botnet performance.

Besides the aforementioned performance metrics, another important metric is scalability for which we simulated two botnets with 200 and 2000 nodes, respectively. In each botnet, a command from the botmaster is published, and every node is designed to locate this command by issuing lookup queries. The simulation ends when all nodes successfully retrieve the command. In the structured botnet case, we ran the modified Kademlia protocol, with k-bucket size $k = 8$ and the number of nodes to ask in parallel $\alpha = 3$. In the unstructured botnet case, we ran the modified Gia protocol, with minimum number of neighbors $min\_nbrs = 3$, maximum number of neighbors $max\_nbrs = 10$ and maximum number of responses $max\_responses = 1$.

Now, we present and discuss the comparison results. For each metric, we first look at the 200-node botnet and then the 2000-node botnet. Figure 5.2 plots the CDFs of the number of hops needed to retrieve a targeted command. In the 200-node botnet, for the structured architecture, 97% of lookups can be completed within 3 hops. The corresponding number for the unstructured botnet is 5 hops. In the 2000-node botnet, despite the increased network size, 99% of lookups under the structured architecture



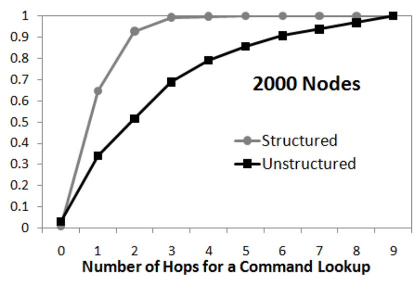
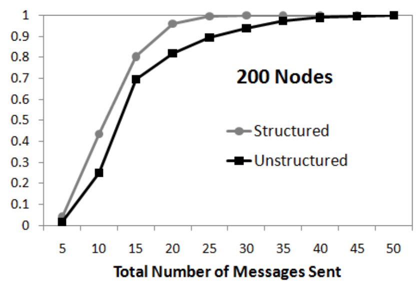


Figure 5.2: CDFs of the number of hops needed for a command-lookup



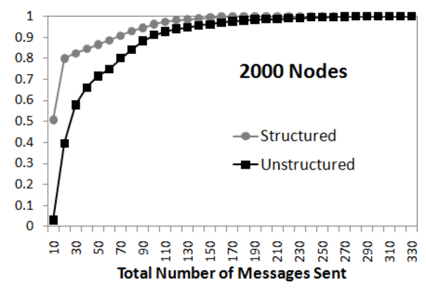
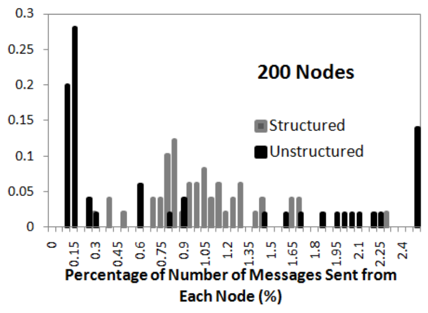


Figure 5.3: CDFs of the total number of messages sent to perform all lookups

are fulfilled within 4 hops, but under the unstructured 8 hops are required. Figure 5.3 shows the CDFs of the total number of SMS messages sent from each node when the command spreads to the entire botnet, which is the total communication overhead. In the 200-node botnet, under the structured architecture, about 80% of nodes generate fewer than 15 messages during the entire period, while under the unstructured architecture 69% of nodes can do so. The average number of messages sent is 11 for the structured and 15 for the unstructured, respectively. In the 2000-node botnet, with more nodes and more lookups, the message overhead unsurprisingly increases. 80% of nodes send fewer than 20 messages (51% of nodes send fewer than 10 messages) for the structured architecture with an average of 22 messages sent by each node. Only 40% of nodes send fewer than 20 messages for the unstructured architecture with an average of 44 messages.

From the above observations, we can see that the structured botnet, in general, requires fewer number of hops to locate a command and incurs a lower message overhead on each node than the unstructured one does in both 200- and 2000-node cases. Compared to the unstructured botnet, the structured architecture also scales better, considering its slight increases in the number of hops and messages when the botnet becomes large. This is expected because in a structured network, data items are placed at deterministic locations so that fewer hops and query messages are required to locate the targeted data and the network can accommodate a large number of nodes. As mentioned before, on smartphones such as Android-based phones, bots are able to send and receive C&C SMS messages stealthily without notifying users. Users may figure that out while seeing the monthly bills, but by then bots have already performed malicious tasks. Even if users are able to see them on the phone, since the C&C messages are disguised as spam, they cause little suspicion. Even so, one may still wonder: would SMSC observe a surge of messages among infected phones and raise alerts? SMS market statistics show that: “In 2009, U.S. cell phone subscribers sent and received on average 390 text messages per month according to the Mobile Business Statistics [21].” We believe that tens of messages overhead per phone may not draw much attention from the SMSC considering a phone's normal messaging volume. Also, since most attacks such as information stealing and spamming are not time-critical, bots do not have to pull commands all at the same time. To further minimize the number of messages sent/received, each bot can be restricted by a threshold. If the number of messages reaches the threshold, the bot will stop sending/receiving messages. The threshold can be customized depending on the usage pattern of SMS on that particular phone. If a bot has frequent normal SMS messaging behavior, (e.g., nearly 3000 texts per US teen per month in Q1 2009 [12]), its threshold of allowing bot communication could be high since this phone is very likely to use a SMS plan and a few blended malicious messages are less noticeable.

Figure 5.4 shows the histograms of load distribution on each node, which is the percentage of total messages each node accounts for during the entire simulation. In



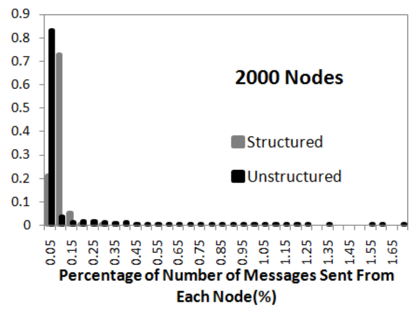


Figure 5.4: Histograms of the percentage of total messages sent from each node

the 200-node botnet, 76% of nodes in the structured botnet each accounts for 0.75% – 1.25% of total messages sent, whereas in the unstructured one, the percentage values are spread out among different nodes ranging from 0.10% to 6%. The average percentage for the structured one is 1.02% and for the unstructured is 1.01%. To gauge the load-balancing more accurately, we calculated a metric defined as: $\sum_{i=1}^{n} |p_i - \overline{p}|$ $(*)$, where n is the total number of nodes, p_i is the load percentage at node i, and $\overline{p}$ is the average percentage across all nodes. The $(*)$ values for the structured and the unstructured are 13.40% and 55.89%, respectively. In the 2000-node case, all nodes' percentages in the structured botnet range from 0.05% to 0.25% while those in the unstructured botnet are distributed within 0.05% – 1.65%, although the average percentages for both the structured and the unstructured are 0.07%. The metric $(*)$ values for the structured and the unstructured are 23.73% and 145.48%. The unstructured case varies more in load distribution leading to poor load-balancing, probably because Gia uses schemes to direct most queries to a few nodes—forming hub nodes.

To estimate the actual delay of locating a command in our mobile botnet, we measured one-hop latency by sending SMS messages between two smartphones. We implemented a SMS send/receive utility on the Android platform and installed it on two G1 phones: one connected to T-mobile and the other to AT&T. The software continually sent out and received SMS messages between two phones and recorded

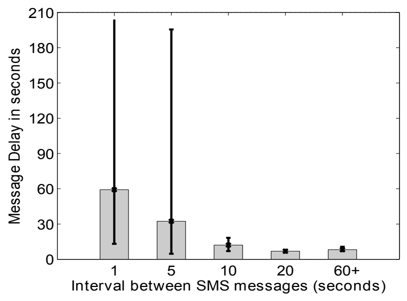


Figure 5.5: SMS message delays

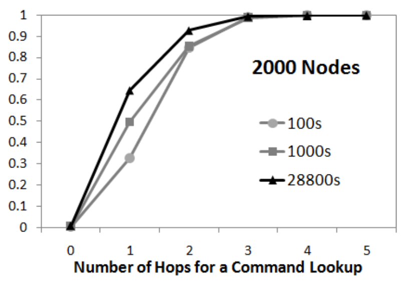


Figure 5.6: CDFs of the number of hops for a command lookup under different mean lifetimes (in sec)

the exact timestamps. The intervals between two consecutive SMS messages were chosen from 1 second to tens of minutes and the message contents were also randomly generated with various lengths to simulate the realistic SMS usage. During the entire experiment, we sent out a total of 138 SMS messages and collected the corresponding message delays, i.e., the difference between the time sending a message from one phone and the time of receiving that message from the other phone. Figure 5.5 depicts min/max/average message delays based on different sending intervals (sending rates). We can see that when SMS messages are sent frequently, the message delays vary a lot and have high average values. Take 1 second as an example. Under this interval, delays range from 15 to 205 seconds with an average of 60 seconds. Similar delay patterns occur when the interval is 5 seconds. The general trend is that as intervals become larger, both delay average and variance drop, and that when the interval is greater than 60 seconds, the delays become stable.

Since mobile attacks such as confidential information stealing (especially related to credit card, account number, etc.) are not time-sensitive, bots can send messages at relatively long intervals to shorten the delay and avoid detection. Using a greater than 1 minute sending interval's delay, we now estimate the total delay for a command-lookup. Under structured Kademlia which uses iterative search, the estimated delay is given by $AverageTotalDelay = 2 \times AverageHops \times AverageOneHopDelay$. When it comes to unstructured Gia which employs recursive search, the equation should be the same. By plugging in the data we obtained, the estimated command-lookup delay is 17 seconds for the structured and 36 seconds for the unstructured in the 2000-node botnet. (To be realistic, we only consider the large-scale scenario.)

The delays seem to be large compared to that of IP-based connections. As briefly mentioned before, our current design does not opt for IP-based C&C or existing IP-based P2P networks for the following reasons. First, some smartphones may not have data plans, not always accessible to the Internet. Second, for smartphones with the Internet access, they can initiate connections to retrieve commands from designated servers but are likely to suffer from a single-point-of-failure. To work in a decentralized P2P fashion, mobile bots should be able to accept incoming connections without any difficulty, which presents a challenge due to private IPs used in most scenarios. A possible solution is to obtain assistance from a third-party such as a mediator server or a rendezvous server, adding complexity to the C&C. Since SMS is ubiquitous across all mobile phones, using SMS as the C&C channel to construct a P2P structure is a feasible and reliable solution for mobile botnets. As future work, we can incorporate IP-based command-transfer into our botnet. For mobile bots without network access, they transmit C&C exclusively via SMS messages. For bots with network access, they can pull commands from an IP-based P2P network. Such a network consisting of PCs can be either constructed by the botmaster or part of an existing P2P network. Doing so may help reduce the message overhead and the delay.

In summary, our simulation results show that the structured architecture outperforms the unstructured one in terms of total number of messages sent, hops needed and delays for a lookup as well as load-balancing, although both the original protocols—Kademlia and Gia—have already been tailored to our mobile botnet's needs through several modifications. Thus, the structured architecture is indeed better suited for our mobile botnet.

5.4.2 Effect of Churn Rates

Now that we have chosen the structured architecture, we would like to see the effect of different mean lifetimes or churn rates on the number of hops for a command lookup, which directly affects the delay of locating a command. To see the trend, in a 2000-node botnet, we varied the mean lifetimes—100s, 1000s and 28800s. The higher the mean lifetime, the lower the churn rate. Presumably, a large mean lifetime indicates a relatively stable network in which fewer steps are needed to locate a command. This assumption is verified in our simulation. We can see that in Figure 5.6, differences, though minimal, exist among the three CDFs. With the mean equal to 100s, the average number of hops is 1.8; with the mean equal to 1000s, the average reduces to 1.7; with the mean equal to 28800s, the average decreases further to 1.4. It turns out that a higher churn rate does not degrade much of the lookup performance.

5.4.3 Can Disguised C&C Messages Go Through?

One concern with our spam-like C&C messages is what if they are filtered and deleted by the service providers without reaching the recipients, which might be the only effective way to mitigate SMS spam (spam-filtering at the end device is not useful as the recipient needs to pay for the messages already). According to some sources [8, 10], mobile carriers do not automatically block SMS spam because there is no spam folder with SMS so that accidental deletion of legitimate messages from the carrier's side cannot be recovered by the users. Also, senders are presumably charged for these messages unlike emails. To confirm this, we ran experiments to see whether carriers will let our spam-like C&C messages pass through. Table 5.1 shows the spam templates for C&C, which are typical spam messages. The random strings highlighted in grey are variables such as passcodes and node IDs. We tried two methods to send them: web-based and smartphone-based. For the first method, we sent all messages twice to an AT&T phone via free texting service at Text4Free.net and txt2day.com

Table 5.1: Spam templates with variable fields in grey

1	Your paypal account was hijacked (Err msg: NzkxMjAzNDlxODExMDUyM183Mz). Respond to http://www.bhocxx.paypal.com using code Q3MDk2NDUyXzEyMzQ1Njc4
2	Free ringtone download at www.myringtone.com, using username VIP, password YTJiNGQxMWw to log on
3	Dear Customer, your order ID dWFuaWRpb3Q is accepted. Please visit: www.xajq.apple.com for more info
4	Your business is greatly appreciated and we would like to award you a free gift. http://www.protending.com/ebay/anVzdDRmdW4
5	To confirm your online bank records, follow the link https://login.personal.wamu.com/logon.asp?id=YWhhaGFoYWg
6	Hey, come on - Purchase G.e.n.e.r.i.c V I A G R A! http://www.WQ9.wesiwhchned.com/default.asp?ID=MTA5MzIxMnc
7	Citi Users: This is an important step in stopping online fraud. Please verify your account at https://www.citi.com.Y2Nzc3Vja3M/verify/
8	Hey alice, I forgot to tell you yesterday that the password to that account(MDkyMzkxMDM0OTgxMjAzN) should be 183MzQyNjIwOTM5XzUxOTQwMTI5
9	Don't miss the chance to win an iPhone 4. Go to www.apple.hak/index.asp?id=OTAxMjc1MjM4OTExMTIzOD, password: QyXzQxNDMyMTg3MzlfNjQ4MTkyMDQ
10	Guess who is tracking your location info? Log on to www.whoistrackingme.com/index.asp?num=YWxqc2hmdy0

respectively. 100% of them reached the designated phone. For the second method, we wrote an application and installed it on an AT&T Samsung Captivate phone running Android OS 2.2. This application automatically sent the spam messages 5 times at different times of a day to another AT&T phone. The application also kept track of whether a message was sent successfully. Out of the 50 messages, 48 messages were sent and delivered to the target phone and 2 messages failed to be sent due to some generic failure at sender's phone that had nothing to do with the carrier. Although we were not able to thoroughly test every possible spam message on different networks, our experimental results were in line with the aforementioned reports and we believe that as few spam-fighting mechanisms are in place, our disguised C&C messages can safely go through the network.

5.4.4 How Do SMS C&C and P2P Structure Become One?

Having an impression of how SMS transmits C&C messages and how a structured P2P topology fits our mobile botnet, one may want to know in detail the way we integrate both into the mobile botnet. We now use a simplified example (Figure 5.7) to illustrate the command publish and search process. For illustration purpose, node IDs and data items' keys are 4-bit long, and SMS messages transmitted are not disguised as spam. In this figure, node 1111 wants to publish certain data—a command—under the key 0111. Note that in Kademlia, data items are stored in nodes whose IDs are close to data items' keys. To locate such nodes, node 1111 first sends SMS messages to nodes in its hard-coded node list; these nodes help to obtain nodes closer to the target from their node lists. The process continues till no closer nodes could be found (this process is omitted in the figure). Finally, node 1111 finds the closest node 0110 ($0110 \oplus 0111 = 0001$), so a publish message containing the command's key (0111), the encrypted command (XXXX) along with a passcode (8888) is sent to node 0110. After verifying the pre-defined passcode

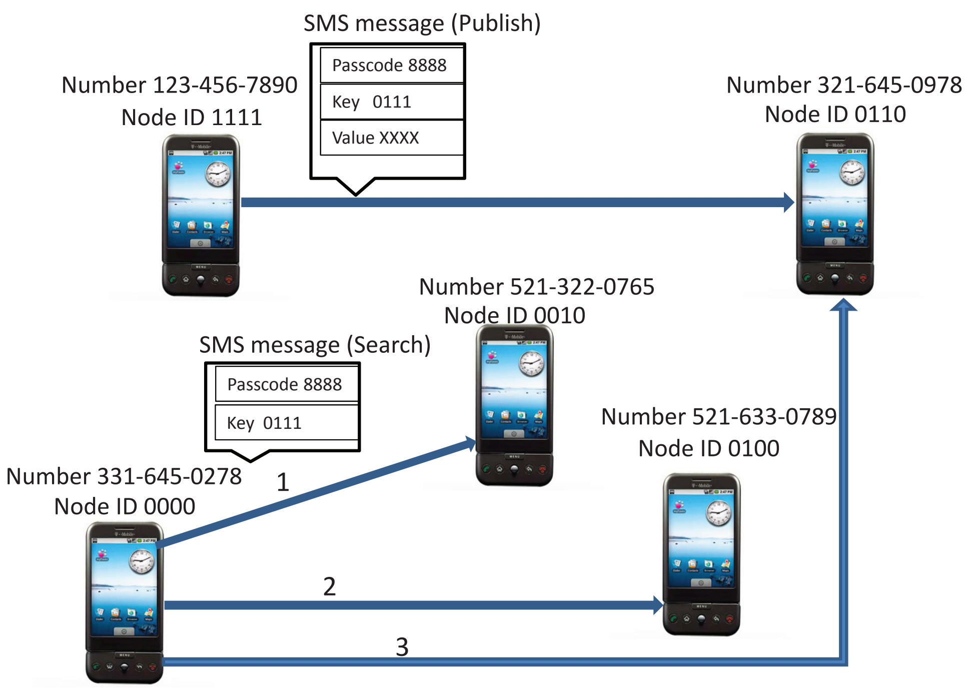


Figure 5.7: Publish and Search

and command, node 0110 stores this information so that later any node requests the command associated with key 0111 it is able to return this command. As for the search process, it is similar to the publish process described. Node 0000 looks up a command associated with key 0111 and it has to find the node whose ID is closer to this key. Node 0000 first asks node 0010; node 0010 points it to node 0100; node 0100 provides the closest one, node 0110. Node 0000 contacts node 0110 to request the command.

5.5 Discussion on Countermeasures

Although we have focused on the design of a stealthy and resilient mobile botnet, we would like to discuss potential defensive strategies against this botnet and challenges in using these techniques.

Similar to the patching mechanism in the PC world, to prevent malicious code from infecting mobile devices by vulnerability exploits, OS vendors and software providers need to push patches to end devices in a timely manner. Certification (only approved applications can be installed) is also an important security measure, but it is far from being perfect as some malware has been able to get around [19] as a disguised harmless application. To nip the mobile malware in the bud, additional protection features are necessary. For example, Kirin [37] is designed for the Android-platform whose certification process is not stringent; it provides application certification at install time using a set of predefined security rules that determine whether the security configuration bundled with an application is safe. With the aid of Kirin, users may be more cautious while installing applications.

Host-based approaches that detect malware at runtime could also serve as a solution. Signature-based detection is effective but cannot handle unknown or polymorphic malware. Therefore, we prefer use of behavior-based detection. The detection scheme proposed in Chapter II that monitors and captures per application behavior can be modified to detect mobile bots. For example, since our bots send SMS messages stealthily without the user's involvement or awareness, the detector could first characterize the normal process of sending SMS messages by a system-call state-diagram and then keep monitoring the system calls that generate outgoing messages to see if there is any deviation from the normal behavior. To detect incoming C&C messages, the detector needs to know the encoding scheme probably through binary analysis so that it can tell which messages are malicious and intercept and delete them before any application's access. However, the botmaster can apply advanced packing and obfuscation techniques to make the binary analysis harder, and periodically update the spam templates as well as the mapping between them and corresponding commands. In addition, host-level detection is susceptible to compromise by the malware, and consumes much resource.

Deploying detection schemes at SMSC is another possible solution. Compared to the host-level detection, this centralized approach can acquire a global view of all phones' SMS activities, although the information of each phone might be limited. As mentioned before, simply filtering out spam will not effectively cut off the botnet's C&C. The reason is that even if carriers dump spam-like SMS messages into a spam folder like email service providers do, spam messages will still reach target phones, stay at a less noticeable place—the spam folder and get commands executed. Blacklisting and SMS sending/receiving rate-limiting may be difficult because our design attempts to minimize the total number of messages sent/received and to balance the load on each bot. As always, matching signatures extracted from known bots' messages can be bypassed by malicious messages with completely new formats or contents. Recall that in Chapter III, the network analyzer searches for botnet-like flow patterns across different hosts based on multiple flow features and identify suspicious hosts accordingly. The same notion can be applied here as well. To differentiate between mobile bots and normal phones, the detector at the SMSC needs to extract distinctive features from SMS traffic patterns. For example, normal phones may have regularities in whom they send messages to and the sending frequency [94]. Utilizing such features, the detector can therefore build normal profiles and identify anomalies. The detector may also adopt a high-level view for detection. The large-scale detection techniques proposed in Chapter IV that exploits structural properties of P2P botnet topologies are applicable to this scenario. Because our mobile botnet utilizes a P2P architecture, the resultant network topology stemmed from SMS activities will be different from that formed by benign phones, given the fact that P2P applications are rare in today's mobile phone networks. The detector at the SMSC can construct communication graphs of mobile devices that send and receive SMS messages through the SMSC and analyze whether the observed graphs have P2P properties.

5.6 Conclusion

As smartphones are getting more powerful, they become potential targets of profit-driven attacks, especially botnets. In this paper, we presented the design of a mobile botnet that utilizes SMS to transmit C&C messages and a P2P structure to construct its topology. Specifically, we used simulation to compare two types of P2P architectures—the structured and the unstructured—based on several metrics critical to the mobile botnet performance. We found that the modified Kademlia—a structured architecture—is more suitable for our botnet in terms of message overhead, delay, and load-balancing. We also investigated possible ways to counter the mobile botnet threat. As future work, we plan to combine SMS-based C&C and IP-based C&C utilizing existing DHT or P2P networks. Since our current work focuses on the aspects of feasibility and efficiency in botnet design, we would also like to measure robustness, i.e., how our botnet performs under different detection and mitigation strategies.

CHAPTER VI

Conclusions and Future Work

Botnets are a lethal weapon for attackers to conduct various cyber crimes. They are increasingly used for spamming, phishing, identity theft and large-scale attacks targeting websites and critical infrastructures. Botnets take a huge toll on governments, businesses as well as individuals, costing them millions of dollars every year. As botnets utilize sophisticated methods such as obfuscated binary code and decentralized C&C structures to hide their presence, detecting botnets is a challenging but critical task.

In this dissertation, we investigated the botnet problem, devised behavior-based botnet detection solutions from a small scale to a large scale at three different levels—the end host, the edge network and the Internet infrastructure, and finally envisioned the direction next-generation botnets are heading to. At the host level, the bot infects machines and tries to spread itself. To mitigate that, in Chapter II, we proposed a containment framework to rate limit the outgoing malicious network traffic as much as possible while minimizing the impact on benign traffic. We developed monitors at the file system, registry and network stack to capture runtime behavior for each process in the system. Depending on how malicious the process is, the framework effectively imposes rate-limiting policies on its traffic. This per-process containment is especially useful for mission or service-critical systems, which cannot afford shutting down the entire system immediately following worm infection but need to maintain operations of critical services and application until they are finished or taken over by healthy systems.

Considering that a host-based approach alone may not be reliable enough because host-resident malware could compromise the detection scheme, we shifted our focus to the local network where bots reside in to see if network-level information would be helpful. We observed that botnets have recently been structured in a decentralized fashion using P2P protocols as opposed to conventional IRC and HTTP for C&C, making detection much difficult. But no matter what C&C botnets use, their underlying properties are the same: bots need to get commands from somewhere and launching attacks thereafter. In addition, they demonstrate anomalous and malicious behavior in the host systems. Based on that, in Chapter III, we utilized the invariant botnets' behavior for detection. We designed a novel hybrid detection framework that incorporates information collected at both host and network levels. This framework is able to detect different types of botnets regardless of their C&C structures with low false alarm rates. One concern about this hybrid detection is its scalability because it requires collection of fine-grained information at the host-level. Since current botnets' sizes are in the order of hundreds of thousands and they are scattered over different networks, to substantially disrupt a botnet, a large-scale detection mechanism is necessary. In Chapter IV, we considered taking a high-level view by exploiting the structural properties of botnets topologies from a graph perspective. Specifically, we measured different network components' capabilities for large-scale P2P botnet detection at the Internet infrastructure level and found that router rendezvous are good venues to deploy detection practices.

The aforementioned three chapters mainly focus on detecting the current botnet threats. In the arms race between attackers and defenders, we were the latter trying to catch up. It is equally or even more important to get ahead of attackers by thinking about new vectors to host botnets and new techniques to control and structure botnets so that countermeasures could be in place by the time the hypothetical botnets become real. Recently, the proliferation of smartphones presents a new and exciting platform to attackers. Envisioning that the next-generation botnets would take advantage of the mobile platform sooner or later, in Chapter V, we presented the design of a proof-of-concept mobile botnet utilizing SMS messages to transmit C&C and P2P as the underlying structure. We compared two P2P topologies via simulations and found that an unstructured topology is more suitable for the mobile botnet. We also showed that the detection solutions proposed in previous chapters can be modified and extended to defend against this mobile botnet threat.

To summarize, the dissertation has made three primary contributions. First, the detection solutions proposed utilize intrinsic and fundamental behavior of botnets without relying on signatures of binaries or packet payloads, so they are immune to malware obfuscation and traffic encryption. Second, the solutions are general enough to identify different types of botnets, not a specific botnet instance. They can also be extended to counter next-generation botnet threats. Third, the detection solutions function at multiple levels—the host, the edge network and the Internet infrastructure—to meet various detection needs. They each take a different perspective but are highly complementary to each other, forming an integrated botnet detection framework.

There are a few future directions that can be pursued following this dissertation.

- **Automatic Behavior Analysis** A key issue in the behavior-based detection is how to quickly and effectively derive distinctive behavior patterns from malicious programs. In this dissertation, manual analysis was used to construct behavior features, which is a common practice nowadays. When the size of the sample pool to be studied is small, manual analysis with human in the loop is tedious but doable. However, the increased use of modularization, polymorphism and metamorphism by malware writers leads to a tremendous surge of new threats, calling for automatic ways to accelerate the analyzing process and save human efforts. This is a challenging task because such automatic systems require efficient monitoring, formalized behavior modeling and extraction, and most importantly, minimal false-positive occurrences.

- **The Implications of Large-Scale Botnet Detection** For actual deployments of large-scale detection mechanisms at the Internet infrastructure, several issues need to be further addressed. First, preprocessing and filtering uninterested network traffic is essential. Multiple techniques may be employed such as port/protocol filtering and flow clustering. Also, in the presence of huge traffic volume, sampling may be necessary and how to do it without losing the big picture remains to be a challenge. Second, since our detection is at the graph level, having no access to fine-grained information, it will also identify regular applications sharing the same topologies as botnets. To avoid misclassification, our approach needs assistance from detection mechanisms at the edge to further confirm that a graph is indeed formed by a botnet. It is therefore necessary to develop such mechanisms for collaboration and communication between the edge and infrastructure networks.

- **Mobile Malware Detection and Mitigation** The design of the proof-of-concept mobile botnet is just the first step; the ultimate goal is to develop detection and mitigation approaches tailored towards smartphones before the threat becomes reality. The detection solutions proposed in this dissertation can be applicable to mobile devices, but modifications and extensions are required. For example, smartphones have resource constraints. A host-based behavioral detection designed for PCs seen in Chapter II might be too heavy-weight for mobile devices. A light-weight solution is desirable. One possibility may be collecting behavior information from devices and getting diagnosis remotely from remote servers or in cloud instead of relying solely on a local detection. Large-scale detection techniques described in Chapter IV need to be customized for use at the SMSC or other vantage points in 3G or 4G cellular networks.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] amule. http://www.amule.org/.

[2] Api hooking revealed. http://www.codeproject.com/system/hooksys.asp.

[3] As names. http://bgp.potaroo.net/cidr/autnums.html.

[4] Change of variables theorem. http://mathworld.wolfram.com/ChangeofVariablesTheorem.html.

[5] China cracks down on sms spam. http://www.redherring.com/Home/19081.

[6] Conficker. http://en.wikipedia.org/wiki/Conficker.

[7] Google yanks over 50 infected apps from android market. http://www.computerworld.com/s/article/9212598 /Google_yanks_over_50_infected_apps_from_Android_Market.

[8] Gsma launches sms spam reporting service. http://www.pcworld.com/businesscenter/article/192469 /gsma_launches_sms_spam_reporting_service.html.

[9] Htc bluetooth vulnerability. http://www.cio.com/article/497146 /HTC_Smartphones_Left_Vulnerable_to_Bluetooth_Attack.

[10] Mobile_phone_spam. http://en.wikipedia.org/wiki/Mobile_phone_spam.

[11] More droiddream details emerge: It was building a mobile botnet. http://www.readwriteweb.com/archives /droiddream_malware_was_going_to_install_more_apps_on_your_phone.php.

[12] More than text: mobile habits of teens. http://www.atelier-us.com/mobile-wireless/article/more-than-text-mobile-habits-of-teens.

[13] Netflow. http://www.manageengine.com/products/netflow/cisco-netflow.html.

[14] Passmark. http://www.passmark.com/.

[15] Prime ssf. https://www.primessf.net/bin/view/Public.

[16] Process explorer. http://technet.microsoft.com/en-us/sysinternals/bb896653.

[17] pvclust package. http://www.is.titech.ac.jp/s̃himo/prog/pvclust/.

[18] Regmon. www.microsoft.com/technet/sysinternals/utilities/regmon.mspx.

[19] Researcher says app store open to malware. http://www.iphonealley.com/current/researcher-says-app-store-open-to-malware.

[20] Rootkitrevealer. http://www.microsoft.com/technet/sysinternals/Utilities /RootkitRevealer.mspx.

[21] Sms market statistics 2009. http://www.massmailsoftware.com /blog/2010/04/sms-market-statistics-2009-know-your-customer/.

[22] Storm worm botnet. http://en.wikipedia.org/wiki/Storm_botnet.

[23] Textfree unlimited. http://itunes.apple.com/us/app/textfree-unlimited-send-text/id305925151?mt=8.

[24] textplus. http://www.textplus.com/.

[25] Waledac botnet. http://en.wikipedia.org/wiki/Waledac_botnet.

[26] Lagrande technology architectural overview. Technical report, Intel Corp., 2003.

[27] Battling botnets for control of computers—microsoft security intelligence report. Technical report, Microsoft Corp., Jan to June 2010.

[28] Messagelabs intelligence: 2010 annual security report. Technical report, Symantec Corp., 2010.

[29] James R. Binkley and Suresh Singh. An algorithm for anomaly-based botnet detection. In *Proceedings of the 2nd conference on Steps to Reducing Unwanted Traffic on the Internet*, 2006.

[30] Abhijit Bose and Kang G.Shin. On mobile viruses exploiting messaging and bluetooth services. In *Proceedings of the 2nd International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2006.

[31] Chih-Chung Chang and Chih-Jen Lin. Libsvm – a library for support vector machines. http://www.csie.ntu.edu.tw/ cjlin/libsvm/.

[32] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutellalike p2p systems scalable. In *Proceedings of ACM SIGCOMM, 2003.*

[33] Shigang Chen and Yong Tang. Slowing down internet worms. In *Proceedings of the 24th International Conference on Distributed Computing Systems*, 2004.

[34] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. Semantics-aware malware detection. In *Proceedings of IEEE Symposium on Security and Privacy*, 2005.

[35] David Dagon, Guofei Gu, Christopher P. Lee, and Wenke Lee. A taxonomy of botnet structures. In *Proceedings of the 23 Annual Computer Security Applications Conference (ACSAC)*, 2007.

[36] Carlton R. Davis, Stephen Neville, Jose M. Fernandez, Jean-Marc Robert, and John McHugh. Structured peer-to-peer overlay networks: Ideal botnets command and control infrastructures? In *Proceedings of 13th European Symposium on Research in Computer Security (ESORICS)*, 2008.

[37] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and Communications Security (CCS)*, 2009.

[38] William Enck, Patrick Traynor, Patrick McDaniel, and Thomas La Porta. Exploiting open functionality in sms-capable cellular networks. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, 2005.

[39] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, and Thomas A. Longstaff. A sense of self for unix processes. 1996.

[40] Shadowserver Foundation. Waledac is storm is waledac? peer-to-peer over http.. http2p? http://www.shadowserver.org/wiki/pmwiki.php/Calendar/20081231, 2008.

[41] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. A virtual machine-based platform for trusted computing. In *Proceedings of the Symposium on Operating Systems Principles*, 2003.

[42] Earl F. Glynn. Correlation "distances" and hierarchical clustering. http://research.stowers-institute.org/efg/R/Visualization/cor-cluster/index.htm, 2005.

[43] Jan Goebel and Thorsten Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[44] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium*, 2008.

[45] Guofei Gu, Junjie Zhang, and Wenke Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*, 2008.

[46] Duc T. Ha, Guanhua Yan, Stephan Eidenbenz, and Hung Q. Ngo. On the effectiveness of structural detection and defense against p2p-based botnets. In *Proceedings of 39th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2009.

[47] Thorsten Holz, Christian Gorecki, Konrad Rieck, and Felix C. Freiling. Measuring and detecting fast-flux service networks. In *Proceedings of 15th Network and Distributed System Security Symposium (NDSS)*, 2008.

[48] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proceedings of the First USENIX Workshop on Large-scale Exploits and Emergent Threats (LEET)*, 2008.

[49] Xin Hu, Matt Knysz, and Kang G. Shin. Rb-seeker: Auto-detection of redirection botnets. In *Proceedings of 16th Annual Network and Distributed System Security Symposium (NDSS)*, 2009.

[50] Jingyu Hua and Kouichi Sakurai. A sms-based mobile botnet using flooding algorithm. In *The 5th Workshop in Information Security and Privacy (WISTP)*, 2011.

[51] Ikee.B. http://www.symantec.com/security_response /writeup.jsp?docid=2009-112217-4458-99.

[52] Marios Iliofotou, Hyun chul Kim, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, and George Varghese. Graph-based p2p traffic classification at the internet backbone. In *Proceedings of 12th IEEE Global Internet Symposium*, 2009.

[53] Marios Iliofotou, Prashanth Pappu, Michalis Faloutsos, Michael Mitzenmacher, Sumeet Singh, and George Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *Proceedings of Internet Measurement Conference (IMC)*, 2007.

[54] Mark Jelasity and Vilmos Bilicki. Towards automated detection of peer-to-peer botnets: On the limits of local approaches. In *Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET)*, 2009.

[55] Thorsten Joachims. Making large-scale support vector machine learning practical. In *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[56] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of the 1st conference on First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[57] Engin Kirda, Christopher Kruegel, Greg Banks, Giovanni Vigna, and Richard A. Kemmerer. Behavior-based spyware detection. In *Proceedings of the 15th USENIX Security Symposium*, 2006.

[58] Tony Lee and Jigar J. Mody. Behavioral classification. Technical report, Microsoft Corp., 2006.

[59] H. Lin, C. Lin, and R. Weng. A note on platt's probabilistic outputs for support vector machines, 2003.

[60] Carl Livadas, Bob Walsh, David Lapsley, and Tim Strayer. Using machine learning techniques to identify botnet traffic. In *Proceedings of the 2nd IEEE Local Computer Networks Workshop*, 2006.

[61] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *Proceedings of the First International Workshop on Peer-to-Peer Systems (IPTPS)*, 2002.

[62] Timothy N. McPherson and Michael J. Brown. Estimating daytime and nighttime population distributions in u.s. cities for emergency response activities. The American Meteorological Society.

[63] Collin Mulliner and Charlie Miller. Fuzzing the phone in your phone. In *Black-Hat Security Conference*, 2009.

[64] Collin Mulliner and Jean-Pierre Seifert. Rise of the ibots: 0wning a telco network. In *The 5th IEEE International Conference on Malicious and Unwanted Software (MALWARE)*, 2010.

[65] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *Proceedings of 19th USENIX Security Symposium*, 2010.

[66] Antonio Nappa, Aristide Fattori, Marco Balduzzi, Matteo Dell'Amico, and Lorenzo Cavallaro. Take a deep breath: A stealthy, resilient and cost-effective botnet using skype. In *Proceedings of 7th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2010.

[67] Jose Nazario. Walking waledac. http://asert.arbornetworks.com /2009/01/walking-waledec/, 2009.

[68] Jose Nazario and Thorsten Holz. As the net churns: Fast-flux botnet observations. In *Proceedings of 3rd International Conference on Malicious and Unwanted Software (MALWARE)*, 2008.

[69] OverSim. http://www.oversim.org/.

[70] Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. A multi-perspective analysis of the storm(peacomm)worm. Technical report, SRI, 2007.

[71] Jian Qiu and Lixin Gao. As path inference by exploiting known as paths. In *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, 2005.

[72] Radmilo Racic, Denys Ma, and Hao Chen. Exploiting mms vulnerabilities to stealthily exhaust mobile phone's battery. In *Proceedings of the 2nd International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2006.

[73] Amir Hassan Rasti, Reza Rejaie, and Walter Willinger. Characterizing the global impact of p2p overlays on the as-level underlay. In *Proceedings of the 11th Passive and Active Measurement Conference (PAM)*, 2010.

[74] Stuart E. Schechter, Jaeyeon Jung, and Arthur W. Berger. Fast detection of scanning worm infections. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2004.

[75] R. Sekar, M. Bendre, P. Bollineni, and D. Dhurjati. A fast automaton-based method for detecting anomalous program behaviors. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2001.

[76] Vyas Sekar, Yinglian Xie, Michael K. Reiter, and Hui Zhang. A multi-resolution approach for worm detection and containment. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*, 2006.

[77] Kapil Singh, Samrit Sangal, Nehil Jain, Patrick Traynor, and Wenke Lee. Evaluating bluetooth as a medium for botnet command and control. In *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2010.

[78] Kapil Singh, Abhinav Srivastava, Jonathon Giffin, and Wenke Lee. Evaluating email's feasibility for botnet command and control. In *Proceedings of 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2008.

[79] SMS. http://en.wikipedia.org/wiki/SMS.

[80] Anil Somayaji and Stephanie Forrest. Automated response using system-call delays. In *Proceedings of the 9th USENIX Security Symposium*, 2000.

[81] Guenther Starnberger, Christopher Kruegel, and Engin Kirda. Overbot - a botnet protocol based on kademlia. In *Proceedings of 4th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2008.

[82] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM 2001.*

[83] W. Timothy Strayer, Robert Walsh, Carl Livadas, and David Lapsley. Detecting botnets with tight command and control. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, 2006.

[84] Symantec. Symantec internet security threat report highlights rise in threats to confidential information. www.symantec.com/press/2005/n050321.html, 2005.

[85] SymbOS.Exy.A. http://www.symantec.com/security_response /writeup.jsp?docid=2009-022010-4100-99.

[86] Patrick Traynor, Michael Lin, Machigar Ongtang, Vikhyath Rao, Trent Jaeger, Thomas La Porta, and Patrick McDaniel. On cellular botnets: Measuring the impact of malicious devices on a cellular network core. In *Proceedings of the 12th ACM Conference on Computer and Communications Security (CCS)*, 2009.

[87] Vladimir Vapnik. *The Nature of Statistical Learning Theory.* Springer, 1995.

[88] Hao Wang, Somesh Jha, and Vinod Ganapathy. Netspy: Automatic generation of spyware signatures for nids. In *Proceedings of Annual Computer Security Applications Conference (ACSAC)*, 2006.

[89] Ping Wang, Sherri Sparks, and Cliff C. Zou. An advanced hybrid peer-to-peer botnet. In *Proceedings of First Workshop on Hot Topics in Understanding Botnets (HotBots)*, 2007.

[90] Ping Wang, Lei Wu, Baber Aslam, and Cliff C. Zou. A systematic study on peer-to-peer botnets. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN)*, 2009.

[91] Roman Waupotitsch, Stephan Eidenbenz, James P. Smith, and Lukas Kroc. Multi-scale integrated information and telecommunications system (miits): First results from a large-scale end-to-end network simulator. In *Proceedings of the Winter Simulation Conference*, 2006.

[92] Georgia Weidman. Transparent botnet command and control for smartphones over sms. In *Shmoocon 2011.*

[93] Matthew M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Proceedings of the 18th Annual Computer Security Applications Conference (ACSAC)*, 2002.

[94] Guanhua Yan, Stephan Eidenbenz, and Emanuele Galli. Sms-watchdog: Profiling social behaviors of sms users for anomaly detection. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2009.

[95] Guanhua Yan, Stephan Eidenbenz, Sunil Thulasidasan, Pallab Datta, and Venkatesh Ramaswamy. Criticality analysis of internet infrastructure. *Computer Networks*, 54(7), 2010.

[96] Ting-Fang Yen and Michael K. Reiter. Traffic aggregation for malware detection. In *Proceedings of the 5th GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2008.

[97] Yuanyuan Zeng, Xin Hu, Abhijit Bose, Haixiong Wang, and Kang G. Shin. Containment of network worms via per-process rate-limiting. In *Proceedings of 4th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2008.

[98] Yuanyuan Zeng, Xin Hu, and Kang G. Shin. Detection of botnets using combined host- and network-level information. In *Proceedings of 40th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2010.