CORRELATION-BASED BOTNET DETECTION IN ENTERPRISE NETWORKS

A Thesis
Presented to
The Academic Faculty

by

Guofei Gu

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Computing

Georgia Institute of Technology
August 2008

CORRELATION-BASED BOTNET DETECTION IN ENTERPRISE NETWORKS

Approved by:

Dr. Wenke Lee, Advisor
College of Computing
Georgia Institute of Technology

Dr. Jonathon Giffin
College of Computing
Georgia Institute of Technology

Dr. Mustaque Ahamad
College of Computing
Georgia Institute of Technology

Dr. Chuanyi Ji
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Nick Feamster
College of Computing
Georgia Institute of Technology

Date Approved: July 3, 2008

To my dear family:

Thank you for all of your love, support and encouragements.

ACKNOWLEDGEMENTS

During my Ph.D. study, I have been extremely lucky to have support, encouragement, and inspiration from many people; without them, this work would not have been possible.

My sincere and deep gratitude goes to my advisor, Dr. Wenke Lee, for his kind guidance and consistent support. He led me on this journey, guided me towards the right direction, influenced me as an active thinker, and provided me with insightful and inspiring advice throughout my entire Ph.D. study. He has had an enormous impact on my professional development, and I hope to guide my students as well as he has guided me.

I would also like to thank other members of my committee, Dr. Mustaque Ahamad, Dr. Nick Feamster, Dr. Jonathon Giffin, and Dr. Chuanyi Ji, for their interest in my work. Their insightful comments have significantly improved the quality of my work.

I have also been very fortunate to be a member of a great security research team in our lab at Georgia Tech. I wish to thank Xinzhou Qin, David Dagon, Yi-an Huang, Prahlad Fogla, Monirul Sharif, Oleg Kolesnikov, Mohamad Kone, Takehiro Takahashi, Roberto Perdisci, Bryan Payne, Paul Royal, Kapil Singh, Junjie Zhang, Robert Edmonds, Abhinav Srivastava, Manos Antonakakis, Martim Carbone, Artem Dinaburg, Long Lu, Ying Li, Jennifer Stoll, Ikpeme Erete, and Claudio Mazzariello, for their valuable assistance on my research and for their contribution to the wonderful and enjoyable graduate experience I have had at Tech. They have enriched my life at Georgia Tech and made it happy and memorable. Their friendship has been my best fortune.

I would also like to thank Phillip Porras and Vinod Yegneswaran for mentoring me during my summer internship at SRI International.

I would particularly like to thank my family. Always supportive and encouraging, my parents are my foundation. I know I owe them my life. Finally, there are no words to

express my love and appreciation to my wife, Jianli. Her exceptional kindness and love makes everything worthwhile.

TABLE OF CONTENTS

DEDICATION iii

ACKNOWLEDGEMENTS iv

LIST OF TABLES x

LIST OF FIGURES xi

SUMMARY xiii

I INTRODUCTION 1

1.1 Botnets: Current Largest Security Threat 2

1.2 Botnet Detection: Research Challenges and Our Goals 6

1.3 Solution Overview 8

1.4 Thesis Contribution and Organization 12

II RELATED WORK 14

2.1 Intrusion and Malware Detection 14

2.2 Alert Correlation and IDS Cooperation 18

2.3 Botnet Measurement and Honeypot-based Tracking 21

2.4 Existing Work on Botnet Detection 22

2.5 A Taxonomy of Botnet Detection Techniques 24

III BOTHUNTER: DIALOG CORRELATION-BASED BOTNET DETECTION . 28

3.1 Bot Infection Dialog Model 30

3.1.1 Understanding Bot Infection Sequences 30

3.1.2 Modeling the Infection Dialog Process 32

3.2 BotHunter: System Design and Implementation 35

3.2.1 A Multiple-Sensor Approach to Gathering Infection Evidence . . 37

3.2.2 Dialog-based IDS Correlation Engine 44

3.3 Evaluating Detection Performance 48

3.3.1 Experiments in an *In situ* Virtual Network 48

3.3.2 SRI Honeynet Experiments 50

3.3.3 An Example Detection in a Live Deployment 54
3.3.4 Experiments in a University Campus Network 55
3.3.5 Experiments in an Institutional Laboratory 57
3.4 Discussion 58
3.5 BotHunter Internet Distribution 60
3.6 Summary 61
IV BOTSNIFFER: SPATIAL-TEMPORAL CORRELATION-BASED BOTNET DETECTION 63
4.1 Botnet C&C and Spatial-Temporal Correlation 65
4.1.1 Case Study of Botnet C&C 66
4.1.2 Botnet C&C: Spatial-Temporal Correlation and Similarity 67
4.2 BotSniffer: Architecture and Algorithms 69
4.2.1 Monitor Engine 70
4.2.2 Correlation Engine 72
4.2.3 Single Client C&C Detection Under Certain Conditions 80
4.3 Experimental Evaluation 81
4.3.1 Datasets 81
4.3.2 Experimental Results and Analysis 84
4.4 Discussion 87
4.4.1 Possible Evasions and Solutions 87
4.4.2 Improvements to BotSniffer 90
4.5 Summary 91
V BOTMINER: HORIZONTAL CORRELATION-BASED, PROTOCOL- AND STRUCTURE-INDEPENDENT BOTNET DETECTION 92
5.1 Problem Statement and Assumptions 95
5.2 BotMiner: Architecture, Design and Implementation 97
5.2.1 BotMiner Architecture 97
5.2.2 Traffic Monitors 99
5.2.3 C-plane Clustering 101

5.2.4 A-plane Clustering . . . 107
5.2.5 Cross-plane Correlation . . . 109
5.3 Experiments . . . 111
5.3.1 Experiment Setup and Data Collection . . . 111
5.3.2 Evaluation Results . . . 114
5.4 Limitations and Potential Solutions . . . 117
5.4.1 Evading C-plane Monitoring and Clustering . . . 117
5.4.2 Evading A-plane Monitoring and Clustering . . . 119
5.4.3 Evading Cross-plane Analysis . . . 119
5.5 Summary . . . 120
VI BOTPROBE: CAUSE-EFFECT CORRELATION-BASED BOTNET C&C DETECTION USING ACTIVE TECHNIQUES . . . 121
6.1 Problem Statement and Assumptions . . . 124
6.2 Active Botnet Probing: Architecture and Algorithms . . . 127
6.2.1 Architecture Design . . . 127
6.2.2 Design Choices of Active Probing Techniques . . . 129
6.2.3 Algorithm Design for Botnet Detection Using Active Probing . . . 132
6.2.4 Evaluating User Disturbance and Detection Accuracy Tradeoff . . . 136
6.3 Experiments with BotProbe . . . 137
6.3.1 BotProbe: a Prototype Active Botnet Probing System . . . 137
6.3.2 In Situ Experimental Evaluation . . . 139
6.3.3 User Study on Normal Chat Probing . . . 144
6.4 Discussion . . . 146
6.4.1 Legal Concerns . . . 146
6.4.2 Limitations and Potential Solutions . . . 147
6.5 Summary . . . 149
VII LESSONS LEARNED AND A FUTURE BOTNET DETECTION SYSTEM . 150
7.1 Summary of Bot* Systems . . . 150
7.2 Lessons Learned . . . 152

7.3 Combining Multiple Techniques in a Future Botnet Detection System . . 154
VIII CONCLUSION AND FUTURE WORK 158
8.1 Conclusion 158
8.2 Future work 160
REFERENCES 162
VITA 172

LIST OF TABLES

1 Selective well-known botnets in history. 3

2 Performance comparison of 1-gram PAYL and SLADE. 43

3 Estimated regression coefficients as initial weighting. 46

4 BotHunter detection and dialog summary of virtual network infections. . . 50

5 Dialog warnings (raw alerts) of BotHunter in 4-month operation in CoC network. 56

6 Normal traces statistics (left part) and detection results (right columns) in BotSniffer evaluation. 82

7 Botnet traces statistics and detection results in BotSniffer evaluation. 83

8 Collected botnet traces in BotMiner evaluation, covering IRC, HTTP and P2P based botnets. Storm and Nugache share the same file, so the statistics of the whole file are reported. 112

9 C-plane traffic statistics, basic results of filtering, and C-flows in BotMiner evaluation. 114

10 C-plane and A-plane clustering results in BotMiner evaluation. 115

11 Botnet detection results using BotMiner. 116

12 BotProbe test on Spybot and Rbot. 141

13 User study of performing P1 and P2 probing, using Interleaved-Binary-Response-Hypothesis algorithm. Most users are detected as normal in two or three rounds. 145

14 Summary of our Bot* detection systems. 150

LIST OF FIGURES

1 Botnet: a typical structure. 2

2 Our correlation-based botnet detection framework. 8

3 Simplified bot infection life cycle. 26

4 Phatbot infection dialog summary. 31

5 Bot infection dialog model. 33

6 BotHunter system architecture. 36

7 BotHunter network dialog correlation matrix. 45

8 Scoring plot from expectation table. 46

9 Scoring plot: 2019 real bot infections. 47

10 Honeynet interaction summary for W32/IRCBot-TO. 52

11 Corresponding BotHunter profile for W32/IRCBot-TO. 53

12 Extended bot infection dialog model. 60

13 Centralized botnet command and control: two representative styles and an IRC-based example. 66

14 Spatial-temporal correlation and similarity in bot responses (message response and activity response). 69

15 BotSniffer architecture. 70

16 $\theta(q)$, the probability of crowd homogeneity with q responding clients, and threshold t. 77

17 Probability of two independent users typing similar length of messages. . . 78

18 $E[N|H_1]$, the expected number of crowd rounds in case of a botnet (vary $\theta_0(2)$, q, α and fix $\beta = 0.01$). 80

19 Possible C&C structures of a botnet: (a) centralized; (b) peer-to-peer. . . . 96

20 Architecture overview of our BotMiner detection framework. 98

21 C-plane clustering. 101

22 Visit pattern (shown in distribution) to `Google` from a randomly chosen normal client. 103

23 Scaled visit pattern (shown in distribution) to `Google` for the same client in Figure 22. 104

24 Two-step clustering of C-flows. 105

25 A-plane clustering. 107

26 Example of hierarchical clustering for botnet detection. 111

27 Two-layer architecture of using active techniques for identifying botnet C&Cs. 127

28 Example active probing techniques. Here cmd' means a modified command packet, seq' means modification is needed on the sequence/acknowledge number to keep the TCP session. 130

29 Disturbance to normal user and the effect on detection. 137

30 Click configuration for BotProbe. The figure shows a configuration for black-box testing on existing bot binaries. If BotProbe is deployed as a middlebox into a real network, we can remove the IRC Server, SimpleResponder, and DNSResponder elements. 138

31 Example combination of multiple techniques in a future botnet detection system. 154

SUMMARY

Most of the attacks and fraudulent activities on the Internet are carried out by malware. In particular, botnets, as state-of-the-art malware, have become a primary "platform" for attacks on the Internet. A botnet is a network of compromised computers (i.e., bots) that are under the control of an attacker (i.e., a botmaster) through some command and control (C&C) channel. It typically contains tens to hundreds of thousands of bots, but some even had several millions of bots. Botnets are now used for distributed denial-of-service (DDoS) attacks, spam, phishing, information theft, distributing other malware, etc. With the magnitude and the potency of attacks afforded by their combined bandwidth and processing power, botnets are now considered as the *largest* threat to Internet security.

Counteracting this emerging threat requires better detection techniques that identify botnets (bots and/or their C&C servers) so that we can mitigate their damage and defend against them. In this thesis, we focus on addressing the botnet detection problem in an enterprise-like network environment. We present a correlation-based framework for botnet detection consisting of detection technologies already demonstrated in four systems: BotHunter, BotSniffer, BotMiner, and BotProbe. Among these systems, BotHunter and BotProbe focus on detecting the *individual* behavior of bots, while BotSniffer and BotMiner are targeted to detect the *group* behavior of bots. A common thread of these systems is correlation analysis, i.e., vertical (dialog) correlation, horizontal correlation, and cause-effect correlation. BotHunter presents *vertical correlation*, or *dialog correlation*, a new kind of network perimeter monitoring strategy that examines the behavior of each *distinct* internal host and focuses on recognizing the infection and coordination *dialog* occurring

during a successful malware infection. BotSniffer and BotMiner present another complementary network perimeter monitoring strategy, i.e., *horizontal correlation*, which focuses on recognizing behavioral similarities and correlations *across multiple hosts*. BotSniffer is designed mainly to capture *centralized* botnet C&C channels using multiple rounds of spatial-temporal correlation, while BotMiner provides a more *general* framework for protocol- and structure-independent botnet detection using clustering analysis of network traffic; thus, BotMiner can be effective even when botnets change their C&C techniques (e.g., protocols and structures). Finally, different from the above *passive* monitoring strategies, which usually require a relatively long time to observe several rounds/stages of botnet communications/activities, BotProbe uses *active botnet probing* techniques in a middlebox to achieve sufficient confidence of a *cause-effect correlation* caused by the command-response pattern of botnet C&C, and only requires observing *at most one* round of actual C&C interaction. In short, we build a comprehensive correlation-based framework for multi-perspective botnet detection, and implement different correlation techniques in different systems that complement each other.

All these Bot* systems have been evaluated in live networks and/or real-world network traces. The evaluation results show that they can accurately detect real-world botnets for their desired detection purposes with a very low false positive rate. These systems are already starting to make an impact in the real world. For example, BotHunter is available to the public at `http://www.cyber-ta.org/BotHunter/`, and in the first five months after its public release, it amassed more than 6,000 downloads.

We find that correlation analysis techniques are of particular value for detecting advanced malware such as botnets. Dialog correlation can be effective as long as malware infections need multiple stages. Horizontal correlation can be effective as long as malware tends to be distributed and coordinated. In addition, active techniques can greatly complement passive approaches, if carefully used. We believe our experience and lessons are of great benefit to future malware detection.

CHAPTER I

INTRODUCTION

During the last few decades, we have witnessed the explosive rise of the Internet and the applications based on it to the point at which they have become an integral part of our lives. While providing tremendous convenience, the growing reliance on the Internet also presents a number of great security challenges. Internet security thereby has become more and more important to those who use the Internet for work, business, entertainment, or education.

Most of the attacks and fraudulent activities on the Internet are carried out by malicious software, i.e., malware, which includes viruses, trojan, worms, spyware, and recently botnets. Such malware has risen to become a primary source of most of the scanning [94], distributed denial-of-service (DDoS) activities [71], direct attacks [11], and fraudulent activities [25, 63, 84] taking place across the Internet. These Internet malware keeps evolving in their forms, e.g., from worms to botnets. Among all the forms of malware, botnets, in particular, have recently distinguished themselves as the primary "platform" [63] on which cyber criminals create global cooperative networks to support the ongoing growth of criminal attacks and activities such as DDoS, spam, phishing, and information theft.

In this chapter, we first introduce the botnet problem and explain why it is a serious security threat. We then outline the research challenges for botnet detection, which is essential for further botnet mitigation and defense, and clarify the goals we want to achieve in our solution. Next, we provide an overview of our solution: a correlation-based framework and the Bot* series of systems for botnet detection. Finally, we present the contributions of the thesis and the organization of the remaining chapters.

1.1 Botnets: Current Largest Security Threat

We begin with the definitions of "bot" and "botnet," the key words of the thesis. A bot is a software robot, or more precisely in the context of our research, a malware instance that runs autonomously and automatically on a compromised machine without the user's consent. The bot code is usually professionally written by some (funded) criminal groups and includes a rich set of supported functionalities [16] to carry out many malicious attacks and activities. Sometimes, we may also use the term "bot" to refer to the bot-infected computer (or, "zombie" computer). A botnet (short for robot network) is essentially a network of bots that are under the control of an attacker (usually referred to as "botmaster" or "botherder"). In the remainder of the thesis, we more formally define a botnet as "a coordinated group of malware instances (bots) that are controlled by a botmaster via some command and control (C&C) channel."[1] We may also refer to a botnet as "a bot army." Figure 1 illustrates a typical structure of a botnet.

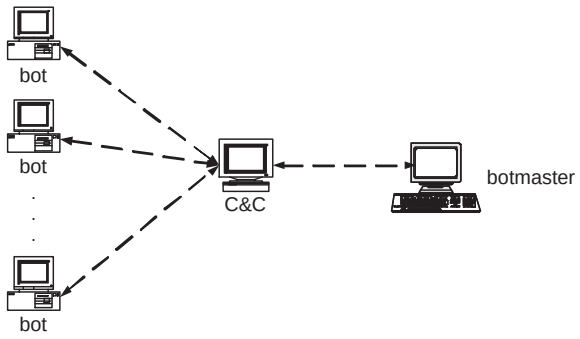


Figure 1: Botnet: a typical structure.

As a state-of-the-art malware form, a bot typically uses a combination of existing advanced malware techniques. For example, a bot can use keylogger techniques (to record a user's keyboard input such as password) and rootkit techniques (to hide its presence in the system). In addition, like the previous generation of malware such as worms, a bot can self-propagate on the Internet to increase the size of the bot army, e.g., infect remote vulnerable

[1] We will revisit this definition in Chapter 5.

hosts through direct exploitation, or propagate through social engineering approaches such as email and instant message. Recently, in one emerging trend, botmasters use compromised Web servers to infect those who visit the websites through drive-by download [81]. By using multiple propagation vectors, the botmaster can recruit many victims. Currently, a botnet typically contains tens to hundreds of thousands of bots, but some had several millions of bots [61].

All bots distinguish themselves from the previous malware forms by their ability to establish a command and control (C&C) channel through which they can be updated and directed by a botmaster. Once collectively under the control of a botmaster, bots form a *botnet*. To control the bot army, a botmaster can use several control mechanisms in terms of protocols and structures. The Internet Relay Chat (IRC) protocol is the earliest, and still the most widely used C&C channel at present. HTTP is also used because Web traffic is generally allowed in most networks. Although centralized control was very successful in the past, botmasters are also exploring distributed control in order to avoid the single point of failure problem. For example, they can use a peer-to-peer structure to organize and control a bot army [44, 52, 64, 106, 112]. We will discuss other related background on botnet C&C mechanisms when we present our detection systems in later chapters.

Within the last decade, many infamous botnets have attracted considerable media coverage. Table 1 lists several examples of such well-known botnets and briefly describes their features.

Table 1: Selective well-known botnets in history.

Date	Name	C&C Protocol	Structure	Distinguishing Description
04/1998	GTbot	IRC	Centralized	First widely spreading IRC bot using mIRC executables and scripts
04/2002	SDbot	IRC	Centralized	First stand-alone and open-source IRC bot
10/2002	Agobot	IRC	Centralized	Very robust, flexible, and modular design
04/2003	Spybot	IRC	Centralized	Extensive feature set based on Agobot
2004	Rbot/rxbot	IRC	Centralized	SDbot descendant, code base wildly distributed
03/2004	Phatbot	WASTE	P2P	Experimental P2P bot using WASTE protocol
05/2004	Bobax	HTTP	Centralized	First well-known spambot using HTTP as C&C
04/2006	Nugache	Self-defined	P2P	First “practical” P2P bot connecting to predefined peers
01/2007	Storm	Kademlia	P2P	Famous large-scale P2P botnet mainly used to send spam
04/2008	Kraken	Self-defined	Centralized	Large botnet penetrating into at least 50 of the Fortune 500 companies

Unlike previous malware such as worms, which are probably *fun-driven*, botnets are

targeted for real financial gain, i.e., they are truly *profit-driven*. Recently, Vint Cerf, "the father of the Internet," likened the botnet scourge to a pandemic and speculated that up to *a quarter of* all computers on the Internet may be participating in botnet-related activities [116]. The magnitude of bot armies and the potency of attacks afforded by their combined bandwidth and processing power have led to a recognition of botnets as the *largest* threat to Internet security [63]. Currently, botnets are the root cause of many Internet attacks and illicit activities [12, 25, 84], listed as follows.

- DDoS attacks. A bot army can be commanded to launch a targeted, distributed denial-of-service attack against any single Internet system/service (e.g., a website) in an attempt to consume the resources (e.g., bandwidth) of the system so that it cannot properly serve its intended users. Nowadays, *all* DDoS attacks are launched from botnet platforms. Although simple in principle and technique, a DDoS attack is very effective because of the magnitude and the accumulated bandwidth of the botnet, and it is very hard to prevent and defend against. As a recent well-known example, such a DDoS attack was launched against the Estonian government and commercial websites in May 2007 [33].

- Spam production. More than 95% of email on the Internet is spam [114], accounting for several billion spam messages in Internet traffic daily and frustrating, confusing, and annoying e-mail users. *Most* of these spam messages are actually sent from botnets. Although the detailed percentage of spam sent from botnets may vary according to different statistics (e.g., 80% in 2004 [65]), many people believe it accounts for more than 95% now. A number of well-known botnets have been used mainly for sending spam, including Bobax [96], an early spambot using HTTP as its C&C, and Storm worm (a.k.a. Peacomm) [44, 52], another infamous P2P botnet aggressively conducting spam activities.

- Click fraud. A botmaster can easily profit by driving the bot army to click on online ads (i.e., issue HTTP requests for advertiser web pages) for the purpose of personal or commercial gain. For example, as reported by Google Click Quality and Security Teams [32], `Clickbot.A` is a botnet that controls 100,000 machines to execute a low-noise click fraud attack through syndicated search ads. According to ClickForensics (`http://www.clickforensics.com`), click fraud accounted for 27.8% of all pay-per-click advertisements in the first quarter (Q1) of 2008. Botnets are boosting click fraud, e.g., click fraud traffic generated from botnets in Q1 2008 was 8% higher than that in Q4 2007.

- Information theft. Bots are actively used to steal sensitive information such as identities, credit card numbers, passwords, or product keys on a user's local machine. By using keyloggers and screen capture, a bot can also easily steal the password of an online banking account. This is becoming a very serious problem. For example, in 2005, the FBI estimated that botnets caused 20 million dollars in losses and theft [20], "including one scheme that bilked a Midwest financial institution out of millions."

- Phishing. Botnets are widely used to host malicious phishing sites. Criminals typically send out spam messages (e.g., using botnets) to trick users to visit fake phishing sites (usually financial related), so that they can obtain users' sensitive information such as usernames, passwords, and credit card numbers on the real financial websites. The independent research and advisory firm Financial Insights [3] estimated that in 2004, global financial institutions experienced more than $400 million in fraud losses from phishing. U.S. businesses lose an estimated $2 billion a year as their clients become phishing victims.

- Distributing other unwanted software, e.g., adware/spyware. Botnets are naturally a good platform on which a botmaster can distribute many other forms of malware.

According to a recent report [105], a discovered botnet "illegally installed adware on hundreds of thousands of computers in the U.S., including some belonging to the military."

Botnets can launch several other forms of attacks or illicit activities. A detailed analysis of these and the impact of malware, particularly botnets, on the Internet economy can be found in a recent report [7] from OECD (Organization for Economic Co-operation and Development) and APEC (Asia Pacific Economic Co-operation).

1.2 Botnet Detection: Research Challenges and Our Goals

To counter botnet attacks, the current largest security threat, we first need to detect the existence of botnets (bots and/or their C&C servers) within a monitored network in order to effectively mitigate and defend against them. Botnet detection has become a daunting task because botnets, as state-of-the-art malware, present considerable challenges:

- Bots are stealthy on infected machines. Since they are used for long-term purpose to make profits, unlike some previous malware such as viruses and worms, bots usually do not aggressively consume CPU/memory/bandwidth resources nor perform noticeable damage to computers in order to avoid the awareness of the user. They can disable existing anti-virus tools and use rootkit techniques to protect them from being detected at a local host. Thus, a host-based solution may not be very effective. In this thesis, we thereby focus primarily on a network-based solution.

- Bot infection is usually a multi-faceted and multi-phased process, incorporating several computing assets, multiple bidirectional network flows, and different infection stages. Thus, looking at only one specific aspect (or, event, stage), as many existing solutions do, may be not effective; it may lead to more false positives and false negatives. In contrast, looking at multiple aspects is more robust and likely to reveal the big picture of a bot infection.

- Bots are dynamically evolving. For example, they can frequently update their binaries as directed, more quickly than a user updates his anti-virus signature base. Thus, static and signature-based approaches may not be effective.

- Botnets can have a very flexible design of C&C channels. They can use different protocols such as IRC or HTTP. They can encrypt the content (e.g., commands) in a C&C communication. They can even use a different structures to organize and control the bot army, e.g., using P2P techniques. Thus, a solution very specific to a certain botnet C&C *instance* is not desirable.

Because of these challenges, existing techniques such as traditional anti-virus tools cannot sufficiently handle the botnet detection problem. In Chapter 2, we provide a detailed overview of various related work (e.g., intrusion/malware detection, honeypot-based botnet tracking, and existing botnet detection approaches) and further explain why these existing solutions are not adequate for botnet detection.

In this thesis, we propose our network-based solution for botnet detection. When designing the solution, we have the following four goals in mind:

1. Our solution should be guided by sound principles that capture the fundamental invariants of botnet behavior rather than symptoms (e.g., scanning).

2. Our solution should provide complementary techniques and cover multiple stages, dimensions, and perspectives.

3. Our solution should be general and extensible. By general, we mean the solution should not be restricted to a specific botnet instance or dependent on a specific symptom. By extensible, we mean the solution should provide an open and flexible framework that can easily incorporate new user-provided components/plug-ins.

4. Our solution should provide practical prototype systems that can work in real-world networks. By practical, we mean that the detection systems can accurately detect

real-world botnets for their desired detection purposes and have a low false positive rate in real-world normal traffic as well as reasonable resource usage.

1.3 Solution Overview

We propose a *correlation*-based framework for effective network-based botnet detection in an enterprise-like network environment such as a university campus network, a regular enterprise network, or simply a local area network (LAN). Within this framework, we present three different correlation techniques: vertical (dialog) correlation, horizontal correlation, and cause-effect correlation. Based on these techniques, we build four anomaly/behavior-based detection systems: BotHunter [46], BotSniffer [48], BotMiner [45], and BotProbe. These four systems can be deployed at the network edge router to monitor through traffic; they will issue alerts when detecting some internal hosts/groups (indicated by IP addresses) as suspicious bots/botnets.

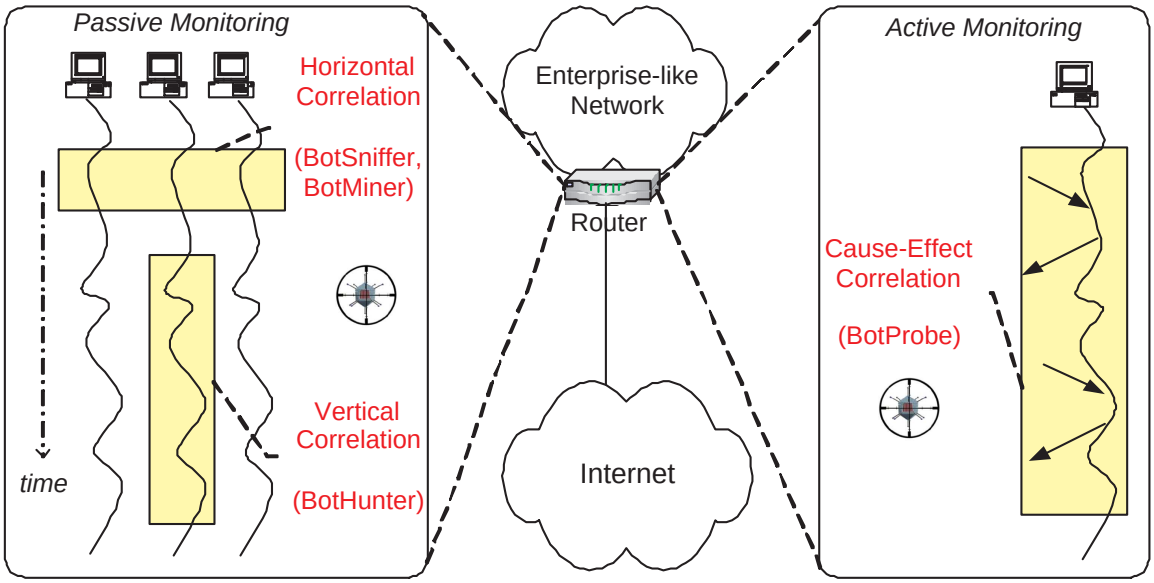


Figure 2: Our correlation-based botnet detection framework.

Figure 2 illustrates our correlation-based botnet detection framework and four detection systems. Among these systems, BotHunter uses vertical correlation, and BotSniffer

and BotMiner use horizontal correlation. While these three systems use a *passive* monitoring strategy, BotProbe uses an *active* monitoring strategy and the cause-effect correlation technique. These Bot* systems have two different detection focuses: BotHunter and BotProbe are used to detect the *individual* behavior of bots, while BotSniffer and BotMiner are used to detect the *group* behavior of a botnet.

BotHunter [46] presents vertical correlation, a new kind of network perimeter monitoring strategy that examines the behavior history of each distinct host. It recognizes a correlated dialog trail (or, evidence trail) consisting of multiple stages and representing a successful bot infection. Therefore, this strategy is also referred to as "dialog correlation." BotHunter is designed to track two-way communication flows between internal assets and external entities, and recognize an evidence trail of data exchanges that match a state-based infection sequence model. BotHunter consists of a correlation engine driven by several malware-focused network detection sensors, each charged with detecting specific stages and aspects of the malware infection process, including inbound scanning, exploit usage, egg downloading, outbound bot coordination dialog, and outbound attack/propagation. The BotHunter correlator then links the dialog trail of inbound intrusion alarms with those outbound communication patterns that are highly indicative of a successful local host infection. When a sequence of evidence matches BotHunter's infection dialog model, a consolidated report is produced to capture all the relevant events and event sources that played a role during the infection process. More details of BotHunter are discussed in Chapter 3.

BotHunter has some limitations. It is restricted to the *predefined* infection life cycle model, and at some stages such as C&C communication, it currently provides only signature-based sensors. Thus, to complement BotHunter, we propose another two systems, BotSniffer [48] and BotMiner [45], which do not necessarily require the observation of multiple *different* stages on an individual host, nor require botnet-specific signatures. Unlike BotHunter's vertical correlation, BotSniffer and BotMiner present another complementary novel network monitoring strategy, "horizontal correlation," which examines

the correlation and similarity *across* multiple hosts. This horizontal correlation technique is inspired by the observation that, because of the pre-programmed activities related to C&C under the control of a botmaster, bots within the same botnet will likely demonstrate spatial-temporal correlation and similarity. For example, they engage in coordinated/similar communication, propagation, and attack and fraudulent activities. However, normal independent hosts (even the previous generation of isolated malware instances) are unlikely to demonstrate such correlated malicious activities. By using horizontal correlation and anomaly detection techniques, both systems do not require *a priori* knowledge of botnets such as captured bot binaries and hence the botnet signatures, and C&C server names/addresses.

BotSniffer [48], designed to detect mainly centralized C&C channels, monitors *multiple* rounds of spatial-temporal correlation and similarity of message/activity responses from a group of hosts that share some common centralized server connection (e.g., IRC or HTTP). Using statistical algorithms, it can achieve theoretical bounds on the false positive and false negative rates within a reasonable detection time (quicker than BotMiner). More details of BotSniffer are discussed in Chapter 4.

An important limitation of BotSniffer is that it is restricted to the detection of botnets mainly using *centralized* C&C channels. BotMiner [45] presents a more *general* detection framework that is independent of botnet C&C protocol and structure. We start from the definition and essential properties of botnets. As defined before, a botnet is a *coordinated group* of *malware* instances that are *controlled* by a botmaster via some C&C channel. The essential properties of a botnet are that the bots communicate with some C&C servers/peers, perform malicious activities, and do so in a similar or correlated way. Accordingly, BotMiner clusters similar communication traffic and similar malicious traffic, and performs cross cluster correlation to identify the hosts that share both similar communication patterns *and* similar malicious activity patterns. Therefore, these hosts are considered bots in the monitored network. More details of BotMiner are discussed in Chapter 5.

While BotHunter, BotSniffer, and BotMiner use a *passive* monitoring strategy, which usually requires a relatively long time to observe multiple stages/rounds of botnet communications/activities, BotProbe uses an *active* monitor strategy to shorten the detection time. It can actively participate in a network session (e.g., a suspicious IRC chatting session), if necessary, by injecting some well-crafted packets to the client within the monitored network. We call this technique "active botnet probing." Our motivation is that, for a large portion of botnet C&C channels (e.g., those using chatting-like protocols such as IRC, which is currently used by *most* of the existing botnets), a C&C interaction has a deterministic command-response pattern. By using active botnet probing in a middlebox, we can gain enough confidence of the *cause-effect correlation* caused by this command-response pattern. Although controversial and clearly limited, BotProbe demonstrates effectiveness on real-word IRC-based botnet detection, and requires observing *at most one* round of C&C interaction. This is very useful in a real-world situation in which IRC botnets are still the majority and their C&C interaction is usually infrequent. More details of BotProbe are discussed in Chapter 6.

Our correlation-based framework and Bot* systems meet our four design goals. We provide a brief explanation here and leave the details for the remaining chapters:

First, each correlation analysis captures some perspective of the invariants of botnet behavior, i.e., infection dialog (dialog correlation) and command-response pattern (cause-effect correlation) within the individual host, and correlated behavior within the group (horizontal correlation). We believe that the idea of correlation analysis can potentially capture the behavior invariants of future malware.

Second, most correlation techniques by themselves involve multiple detection sensors covering different stages/aspects. For example, BotHunter employs several sensors covering multiple different infection stages. In addition, different correlation techniques and detection systems complement each other quite well. For instance, some focus on *individual* bot detection, while some focus on the detection of the *network* of bots. Furthermore,

each system may have its own limitations and coverage. However, when combined, they can complement each other to enlarge the detection coverage from multiple different perspectives. We show the architecture of combining our multiple techniques in a future botnet detection system in Chapter 7.

Third, our framework and systems are general and extensible. In design, they target a certain *class* of botnets. In other words, they are not restricted to a very specific botnet *instance*. Even BotProbe, although it appears to target an IRC botnet, is applicable to a general class of botnets that have deterministic, interactive C&C (e.g., chatting-like communication such as IRC or instant message). In addition, these systems are open and extensible. That is, they are amenable to the adding of new detection sensors. We provide a concrete example in Chapter 3.

Finally, our systems are practical and can work in the real world. This will be discussed in detail throughout the remainder of the thesis.

1.4 Thesis Contribution and Organization

In this thesis, we make the following main contributions:

1. We propose a correlation-based framework for multi-perspective botnet detection. In this framework, we introduce several new and complementary network monitoring and correlation analysis techniques, i.e., vertical (dialog) correlation, horizontal correlation, and cause-effect correlation. Each of these correlation techniques provides a relatively sound principle that captures some fundamental behavior invariant of botnets to guide our anomaly-based detection schemes. While vertical correlation captures the dialog nature in the multi-stage bot infection life cycle, horizontal correlation captures the coordination and similarity nature within the same botnet. The cause-effect correlation captures the non-human driven, deterministic command-response pattern of a certain class of botnet C&C channels. We believe the general principles behind these correlation analysis techniques could also be applicable to

detecting future advanced malware.

2. We provide four practical botnet detection prototype systems (i.e., BotHunter, BotSniffer, BotMiner, and BotProbe) that use our correlation techniques. These systems are evaluated on real-world network traffic and shown to accurately detect botnets with a low false positive rate. Our work is starting to make an impact in the real world. For example, BotHunter, available to the public at `http://www.cyber-ta.org/BotHunter/`, has amassed more than 6,000 downloads in the first five months since its public release.

The remainder of this thesis is organized as follows. Chapter 2 introduces related work and explains why the existing work cannot adequately solve the botnet detection problem. In particular, we present a taxonomy of botnet detection techniques and show the difference and coverage of existing detection approaches. Chapter 3 presents the motivation of dialog correlation and the detailed design, implementation, and evaluation of the BotHunter system. Chapter 4 presents the motivation of using spatial-temporal correlation, and the design, implementation, and evaluation of the BotSniffer system, as well as a discussion on the limitations and further improvement. Chapter 5 introduces the BotMiner system, explains its unique feature of protocol- and structure-independence, presents the design, implementation and evaluation of BotMiner, and then discusses its limitations. Chapter 6 presents BotProbe system, including the motivation, design, implementation and evaluation. Chapter 7 summarizes the lessons learned from the Bot* systems and presents an architecture that combines multiple discussed techniques in a future botnet detection system. Finally, Chapter 8 concludes the thesis and describes directions for future work.

CHAPTER II

RELATED WORK

In the previous chapter, we identified the research challenges for botnet detection. In this chapter, we will answer the following questions: Why are existing techniques not sufficient for botnet detection? How are they related to or different from our solution? In particular, to compare existing solutions, we propose a taxonomy of botnet detection techniques and clarify the coverage of different approaches.

2.1 Intrusion and Malware Detection

Existing intrusion and malware detection techniques can generally be categorized into host-based or network-based solutions. Host-based detection techniques are very important to recognize malware binaries (e.g., viruses) and host-level anomaly behavior (e.g., a certain system call invoked, a certain registry key created). Among these techniques, anti-virus tools are useful for traditional virus detection for a long time [99]. Another typical example of host-based intrusion detection techniques is system call-based monitoring [39]. However, when facing the botnet issue, these purely host-based detection techniques have several problems. First, traditional anti-virus tools are based on signatures and essentially requiring a comprehensive, precise, and frequently updated signature base. However, botnets can easily evade signature-based detection by updating themselves more frequently than users update their signature base. Second, host-based detection systems are at the same privilege level as bots on the same host. Thus, bots can disable anti-virus tools in the system and/or use rootkit techniques to protect themselves from detection at the local host. Actually, as state-of-the-art malware, numerous bots have already used all these tricks. Indeed, the detection rate of bots is relatively low compared to that of traditional malware. For example, Kraken was reported to be undetected by 80% of the commercial

anti-virus tools on the market [62]. In 2007, a study from PandaLabs [4] found that even with correctly installed up-to-date protection (e.g., anti-virus tools), a significant portion of PCs (22.97%) still became infected by malware. Considering the fact that millions or even 1/4 of Internet PCs are related to botnet activities [116], the actual percentage could be higher. Finally, behavior-based host level realtime monitoring usually contributes to significant system performance overhead, so these solutions become even less attractive to normal users.

Therefore, in the scope of this research, we care more about network-based detection solutions and consider host-based techniques orthogonal to our network-based approaches. In the remainder of this chapter, we limit our focus to the most relevant work, mainly network-based studies.

Network-based intrusion detection research has proposed many techniques and systems. Snort [86] and Bro [74] are two representative misuse (or, signature) based intrusion detection systems (IDSs). They rely on a large signature base (which precisely describes what attacks look like) to recognize intrusion attempts in network traffic. The fundamental weakness of these signature-based IDSs, similar to traditional anti-virus tools, is that they cannot detect new attacks because they have never been seen before, and thus have no signatures. Anomaly-based IDSs can solve this limitation by describing what *normal* traffic looks like, and any significant deviation from the normal is considered an *anomaly*. Two examples of such IDSs are PAYL [110, 111] and Anagram [109], which examine the payload of an inbound packet, perform n-gram analysis, and then detect exploits (e.g., shellcode) in the payload. The main weakness of such an anomaly-based solution is that it may cause more false positives.

Prior to the prevalence of botnets, worms were the typical malware form. A worm is essentially a self-propagating malware instance that can replicate itself through network infection (or sometimes through social engineering tricks such as email or instant message). The main difference between worms and botnets is that worms do not have a command

and control (C&C) channel. Thus, botnets are fundamentally more flexible than worms. Another difference could be their motivation. Whereas worms are more likely fun-driven, launched by attackers who want to have fun or show off in the "blackhat" community, botnets are more profit-driven, launched by attackers for profit.

There is numerous work on worm detection. Since worms generally use scanning, which provides a quick and automatic way to propagate [93], almost all of the worm detection approaches focus on the detection of scanning traffic/behavior. Moore [70] proposed the use of distributed "network telescopes" for early warning, i.e., using a reasonably large fraction of dark address space to observe security events such as worm scanning traffic occurring on the Internet. Provos [80] and Dagon et al. [30] proposed to use honeypot techniques to gather and identify worm attacks. Zou et al. [127] proposed a Kalman filter-based detection algorithm, an approach that detects the trend of illegitimate scans to a large unused IP space. Wu et al. [120] proposed a victim counter-based detection algorithm that tracks the increasing rate of newly infected outside victims. Jung et al. [57, 58] and Weaver et al. [115] proposed worm detection approaches based on the observation that scanning worms usually cause a high failed connection ratio. Gu et al. [47] proposed the Destination-Source Correlation (DSC) algorithm for worm detection by using an anomaly detection technique that considers both the infection propagation nature and the scanning activity of worms.

While some of the above existing intrusion and malware detection techniques can be helpful in recognizing some anomaly aspect of botnets, they are not by themselves well-suited for botnet detection for the following reasons:

- Most detection systems focus on examining mainly (or only) *inbound* network traffic for signs of malicious point-to-point intrusion attempts. They have the capacity to detect initial incoming intrusion attempts, and the prolific frequency with which they produce such alarms in operational networks is well documented [91]. However,

distinguishing a *successful* local host infection from the daily myriad scans and intrusion attempts is as critical and challenging a task as any facet of network defense. In addition, because of recent advances in malware, particularly the botnet, it is harder to accurately detect when it initially penetrates the monitored network, as machines can be infected by botnets using many ways other than traditional remote exploitation. For example, an internal user may click on a malicious email attachment and get infected, or a user may visit a website and get infected via drive-by downloads, which is now occurring quite frequently [81]. Moreover, an already infected laptop may be carried in and connected to the monitored network. Such examples have compelled us to develop a detection capability for already compromised machines inside monitored networks regardless of how they have been infected. Therefore, we must monitor both inbound and outbound network traffic instead of just incoming traffic.

- A botnet is very flexible, and its infection life cycle can consist of several different stages and aspects. However, existing approaches examine only some certain symptoms such as scanning, so they are less likely to detect botnets. They can cause false positives if a non-bot machine (probably a normal host or other forms of malware) has scanning-like activities. They can also cause false negatives if a bot does not scan, or although still does scan, it evades the specific scan detection technique. Therefore, we still need new techniques that are more suitable for botnet detection and ideally, that follow the four design goals we have proposed.

Finally, we agree that traditional intrusion and malware intrusion detection techniques are still useful for recognizing certain aspects of botnets. As we show later in our detailed solution in the following chapters, some of these existing techniques can be the building blocks for a new system that combines them with our detection techniques following a systematic correlation strategy.

2.2 Alert Correlation and IDS Cooperation

A significant amount of related work has investigated alert correlation and IDS cooperation techniques that combine multiple alerts, events, or aspects of network intrusion detection. These techniques enable an analyst to obtain higher-level interpretations of network detector alert streams, thereby alleviating noise-level issues with traditional network IDS.

The main purpose of alert correlation is for log reduction, multi-step attack detection, and attack intention recognition. In particular, the techniques used to recognize multi-stage attacks share some similarities with our vertical (dialog) correlation technique used in BotHunter. We address their differences as follows.

One approach to capture complex and multi-step attacks is to explicitly specify the stages, relationships, and ordering among the various constituents of an attack. As an illustration, USTAT [54] and NetSTAT [104], two IDSs based on state transition analysis techniques, specify computer attacks as sequences of actions that cause transitions in the security state of a system. In addition, Valeur et al. [103] performed multi-step attack correlation according to attack scenarios specified *a priori* using STATL [36], a language for expressing attacks as states and transitions. Other such systems are JIGSAW [100], a system that uses notions of concepts and capabilities for modeling complex attacks, and a system proposed by Ning et al. [73], which provides a formal framework for alert correlation, and CAML [22], a language framework for defining and detecting multi-step attack scenarios. Unlike BotHunter, all of these techniques are based on *strict causal relationships*, e.g., pre-conditions and post-conditions, or a *strict temporal sequence* of attacks. One of their obvious limitations is that the dependencies and sequences need to be manually specified *a priori* for all attacks, yet such dependencies/sequences are often unknown or very loose. Moreover, a missing event in the dependences/sequences will fail the entire correlation. With regard to botnet detection, although bot infections do regularly follow a series of general stages, we find it rare to accurately detect all steps, and find it equally difficult to predict the order and time-window in which these events are recorded. Thus, the

above alert correlation techniques are not suitable for botnet detection. In contrast, BotHunter does not have a strict restriction in causal dependences or temporal sequences, and can tolerate missing events during the infection flow.

Another IDS, GrIDS [95], aggregates network activity into causal graphs that can be used for analyzing causal structures and identifying policy violations. Ellis et al. [37] and Jiang et al. [56] describe behavioral-based systems for detecting network worms based on tracking propagation graph. In contrast to the above systems based on a global causal graph or a propagation graph, BotHunter focuses on the problem of bot detection and uses local infection dialog correlation as a means to define the probable set of events that indicate a bot infection.

Sommer et al. [91] described contextual Bro signatures as a means for producing expressive signatures and weeding out false positives. These signatures can capture two dialogs and precisely define multi-step attacks. BotHunter differs due to the requirement that several flows be simultaneously monitored across many participants (e.g., infection source, bot victim, C&C, propagation targets) and that the evidence-trail-based approach loosely specifies bot infections.

Alert correlation modules such as CRIM [26] provide the ability to cluster and correlate similar alerts. The system can extract higher-level correlation rules automatically for the purpose of intention recognition. Alert fusion techniques can greatly reduce log size by clustering similar events under a single label [102]. The similarity is usually based upon either attributing multiple events to a single threat agent or providing a consolidated view of a common set of events that target a single victim. Valdes and Skinner [102] proposed a two-step probabilistic alert correlation based on attack threads and alert fusion. We consider this line of work to be complementary to BotHunter, i.e., these fusion techniques could be integrated into our detection systems as a preprocessing step in a multi-sensor environment.

The main purpose of IDS cooperation is to collect information from multiple sources/IDSs

in order to detect distributed and coordinated attacks such as worm propagation. Such techniques share some similarities with our horizontal correlation techniques used in BotSniffer and BotMiner. We address their differences as follows.

Several work, including EMERALD (Event Monitoring Enabling Responses to Anomalous Live Disturbances) [78], AAFID (Autonomous Agents For Intrusion Detection) [15], DIDS (Distributed Intrusion Detection System) [90], and CARDS (Coordinated Attack Response & Detection System) [123], proposed distributed architectures that combine multiple monitors/detectors/agents for intrusion detection and response capability. These techniques provide a distributed, architectural-level solution for tracking distributed and coordinated attacks across multiple machines. In another related work, Abad et al. [9] proposed to correlate data among different sources/logs (e.g., syslog, firewall, netflow) to improve final intrusion detection accuracy. All these techniques are different from BotSniffer/BotMiner in that they merely provide an abstract, high-level, architectural solution as a general hybrid IDS instead of providing concrete detection algorithms and techniques for concrete attacks such as botnets. However, these architectural solutions could be complementary to our work.

Xie et al. proposed Seurat [122], which can detect aggregated anomalous events such as worm propagation by correlating host file system changes across space and time. Malan [67] proposed to use collaborative groups of machines to exchange summaries of recently executed system calls, in order to detect commonly propagated malware (e.g., worm) within the network. These two approaches are similar to our horizontal correlation used in BotSniffer/BotMiner. However, fundamentally different from BotSniffer/BotMiner, they require deploying a host-based sensor to monitor system file changes or system call sequences on *every* machine in the network. First of all, these host-based monitors can cause significant performance overhead on a local machine. Second, such wide-scale installation on every machine is costly and generally not very realistic. Third, as discussed before, these host-based monitors can be disabled or fooled by advanced bots. BotSniffer and BotMiner,

however, using a network-based solution, avoid the above weaknesses. Although similar in concept, the detailed correlation and detection algorithms also differ. Finally, we note that because their features (file system change and system call sequence) are different from those of BotSniffer/BotMiner, Seurat [122] and the techniques from Malan [67] can potentially complement our solution.

2.3 Botnet Measurement and Honeypot-based Tracking

Much of the research on botnets has focused on gaining a basic understanding of the nature and full potential of the botnet threat, e.g., on the measurement study, collection, and tracking issues.

Measurement studies can help us understand the botnet threat. Cooke et al. [25] conducted several basic studies of botnet dynamics. Dagon et al. [31] proposed using the DNS sinkholing technique for botnet study and pointed out the global diurnal behavior of botnets. Barford and Yegneswaran [16] examined the bot source code to provide an inside look at the botnets. For example, they analyzed the structural similarities, defense mechanisms, and command and control capabilities, of major bot families. Collins et al. [24] presented their observations of the relationship between botnets and scanning/spamming activities.

For effective botnet collection and tracking, researchers commonly use honeypot techniques. Freiling et al. [40], using honeypots to track botnets, provided an early report for understanding the phenomenon of botnets. Nepenthes [13] is a low-interaction honeypot that simulates several vulnerabilities and automates the collection of malware binaries. Rajab et al. [82] conducted a multi-faceted approach to collect bots and track botnets, and provided an in-depth study of current botnet activities. By using honeypot techniques to collect and track botnets, researchers can analyze bot binary and behavior, and then extract signatures for content-based detection or C&C server information for response (e.g., DNS sinkhole [31]).

Although honeypots are effective tools for collecting and tracking botnets, they have

several limitations. First, low-interaction honeypots such as Nepenthes [13] can capture attacks from only a limited number of known exploits that they faithfully emulate, and high-interaction honeypots can neither implement all services nor deal with the problem of scaling. Second, honeypots are mainly designed to capture malware that propagates via scanning for remote vulnerabilities, so they cannot easily capture malware using other propagation methods such as email and Web drive-by download,[1] which are probably two of the most widely used propagation vectors [6, 81]. Third, there is no guarantee on the frequency or volume of malware captured using this approach because a honeypot can only wait and hope for the malware to contact it. Fourth, malware may avoid scanning the networks with "known" honeypots [17], and it can detect virtual machine environments commonly used for honeypots [41, 50, 128] and alter its behavior to evade analysis. Finally, honeypots report infections on only their decoy machines; they generally cannot directly tell which non-decoy machines in the enterprise network are members of a botnet. These weaknesses limit the capability of honeypots as effective *detection* systems.

2.4 Existing Work on Botnet Detection

Botnet detection is a relatively new area. Recently, several papers have proposed various approaches for detecting botnets.

Binkley and Singh [18] proposed combining both IRC statistics and TCP work weight (i.e., anomaly scanning activity) for detecting IRC-based botnets. Their approach is useful only for detecting certain botnet instances, i.e., IRC bots that perform scanning.

Ramachandran et al. [85] proposed using DNSBL (DNS blacklist) counter-intelligence to locate botnet members that generate spam. The basic assumption of this approach is that botmasters may use DNSBL to query the status of their bots, and a machine that queries many others but that is rarely queried by others is suspicious. This heuristic may be useful in some cases, but not generally valid and can cause many false positives. As a result, this

[1] Recent advance of client-side honeypot techniques like HoneyMonkey [113] and web-based honeypots [89] could partially relieve this limitation.

approach is restricted to limited instances of spam botnets.

Rishi [43] is a signature-based IRC botnet detection system that matches known nickname patterns of IRC bots. Similar to a typical signature-based anti-virus tool or IDS, this approach is accurate only if a comprehensive and precise signature base is available, but possesses the inherent weaknesses of signature-based solutions such as its inability to detect bots without using known nickname patterns.

Livadas et al. [66, 98] proposed a machine learning based approach for botnet detection using some general network-level traffic features (e.g., bytes per second) of chatting-like protocols such as IRC. Karasaridis et al. [59] studied network flow level detection of IRC botnet controllers for backbone networks. The above two approaches are similar to our BotMiner's work in C-plane clustering but different in many ways. First, they are used to detect *IRC-based* botnets (by matching a known IRC traffic profile, e.g., low volume, chatting-like, or having a PING-PONG pattern), while BotMiner does not have the assumption of using known C&C traffic profiles. Second, they can detect botnets using only a centralized structure, while BotMiner can detect any C&C structures such as P2P. Third, BotMiner uses a different traffic feature set based on a new communication flow (C-flow) data format instead of traditional network flows. Finally, BotMiner uses full knowledge in both C-plane and A-plane information instead of only (or mainly) C-plane network flow records.

Stinson and Mitchell proposed BotSwat [97], a host-based taint tracking system to identify programs that use received network data (from an untrustworthy external source) in some system call argument without intended local user input or explicit permission (e.g., whitelist), an attempt to identify the *potential* remote control behavior of bots. This approach may cause non-negligible false positives (because many legitimate programs may also use a portion of network traffic in some of their system call arguments) and a performance penalty (taint propagation analysis is very heavy, so it is generally used only for the purpose of analysis instead of detection).

Yen and Reiter proposed TAMD [124], a system that detects *potential* malware (including botnets) by aggregating traffic that shares the same external destination, similar payload, and that involves internal hosts with similar operating systems. The concept of traffic aggregation is similar to the horizontal correlation strategy used in BotSniffer and BotMiner. TAMD's aggregation method based on destination networks focuses on networks that experience an increase in traffic as compared to a historical baseline. In addition, this aggregation method is limited to aggregate bots using a centralized C&C structure (i.e., bots that share a common destination server), so it will likely fail on P2P-based botnets. Different from BotSniffer and BotMiner, which focus on botnet detection, TAMD aims to detect a broader range of potentially suspicious hosts as long as they share the same external destination, similar payloads, and similar OS platforms; hence TAMD could cause a higher false positive rate than BotSniffer/BotMiner. Since the aggregation features of TAMD differ from those of BotSniffer/BotMiner, they can complement one another in botnet and malware detection.

2.5 A Taxonomy of Botnet Detection Techniques

In this thesis, we propose four new botnet detection systems: BotHunter [46], BotSniffer [48], BotMiner [45], and BotProbe. We leave the details of the techniques of each system for the following chapters. Here, to provide the reader with quick and systematic knowledge of the features, relationships, and differences among all existing detection techniques including ours, we present a taxonomy of botnet detection methods from seven dimensions.

Our first dimension is whether the solution is based on host or network. We have already discussed the advantages and disadvantages of each solution. Among botnet detection systems/techniques we have introduced, BotSwat [97] is the only host-based solution; the others, including our Bot* systems, are network-based techniques.

Our second dimension is whether the solution is based on signature or behavior/anomaly.

For this dimension, Rishi [43] is the only signature-based solution; the others, including our Bot* systems, are behavior/anomaly-based techniques. The advantages and disadvantages of both solutions have already been discussed before. We note here, although mainly using behavior/anomaly-based techniques, our Bot* systems have been carefully designed by combining various techniques covering different aspects *within* each system and *across* different systems to complement each other. Therefore, we can achieve a very low false positive rate, as shown in the following chapters.

Our third dimension is whether the solution is passive or active. All the above techniques/systems are passive, i.e., they passively monitor the network traffic or system behavior, except BotProbe, which uses an *active* monitoring strategy and which may participate in a botnet communication by inserting or modifying some traffic, if necessary. Compared with an active strategy, the advantage of a passive one is clear: it is safe because it does not interfere existing communication/activity. However, many such passive solutions may require monitoring a relatively longer time to observe multiple stages/rounds/instances of botnet communications/activities in order for accurate detection. An active approach such as BotProbe can compensate for this limitation. For example, BotProbe only requires observing *at most one* round of C&C interaction. By performing active botnet probing several times, BotProbe can gain enough confidence of a *cause-effect correlation* caused by the command-response pattern of botnet C&C. Nevertheless, an active approach such as BotProbe is still controversial and clearly limited. Thus, our framework includes both passive and active approaches that complement each other.

The fourth dimension we consider is the detection phase. The infection life cycle of a bot has roughly two phases:[2] preparation and operation. In the preparation phase, an innocent host becomes a bot by a remote infection or by the mistaken execution of some malicious executable (e.g., in an email attachment), and prepares to be controlled through C&C. A host starts its preparation phase when it is initially attacked, and completes this

[2]Note, later in Chapter 3, we further split the infection life cycle into more detailed stages.

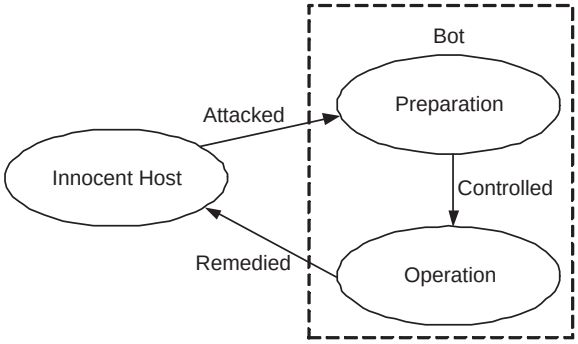


Figure 3: Simplified bot infection life cycle.

phase when the full functional bot binary is executed. When this bot attempts to connect to a C&C channel, it begins its operation phase, when it can be directed by a botmaster to perform any activities. Thus, we can classify a botnet detection technique according to which phase alerts can be issued, during the preparation phase or during the operation phase. All introduced botnet detection systems/techniques, except BotHunter, work only in the operation phase. BotHunter can issue alerts either in the preparation phase (detecting early bots during penetration time) or in the operation phase (detecting bots when they are already there regardless of how they penetrated into the network).

The fifth dimension we consider is the detection target, or, whether the target is an *individual* bot or a *network/group* of bots. Livadas et al. [66, 98], Karasaridis et al. [59], TAMD, BotSniffer, and BotMiner focus on the detection of groups of bots, while others focus on detecting individual bots. These two kinds of solutions are fundamentally complementary. While the group-based approach requires observing more bots (e.g., at least two) for detection, it could discover an anomaly that may not be noticeable at an individual host level. Our framework includes both individual-based (BotHunter and BotProbe) and group-based (BotSniffer and BotMiner) approaches.

Our sixth dimension relates to the detection assumptions, that is, whether a solution requires other out-of-band information (e.g., DNSBL) or not. Among introduced solutions, some (Ramachandran et al. [85]) requires information from other sources such as DNSBL, and some (Karasaridis et al. [59]) requires bootstrapping the clustering analysis from alert

information (e.g., scanning activity) provided by other systems. Except for these two, other techniques/systems require no out-of-band information. They are relatively independent and capable of working directly on network traffic or a host system.

Our final dimension is whether a solution is restricted to or dependent on some certain C&C technique (e.g., protocol and structure). Many existing approaches work for only a certain C&C protocol or structure. Rishi [43] and the approach by Binkley and Singh [18] are designed for only IRC-based botnets. Likewise, approaches by Livadas et al. [66, 98] and Karasaridis et al. [59] are shown to detect IRC botnets in their papers. Their techniques could be used to detect other chatting-like C&C. However, they are still restricted to a centralized structures, and both require knowledge of the C&C traffic profile. TAMD [124] is not restricted to C&C protocols, however, its aggregation by destination is restricted to centralized C&C structures only. Similarly, our BotSniffer focuses on centralized botnet C&C detection. BotSwat [97], the solution by Ramachandran et al. [85], BotHunter and BotMiner are independent of the botnet C&C techniques. However, BotSwat [97] is a host-based solution, and the solution by Ramachandran et al. [85] is fundamentally limited to the detection of specific spambots. In addition, BotHunter is dependent on the infection model. Only BotMiner is a fundamentally general network-based botnet detection framework independent of botnet C&C protocols or structures.

In the following chapters, we will introduce the details of our Bot* systems one by one and then provide a comprehensive summary of the four systems.

CHAPTER III

BOTHUNTER: DIALOG CORRELATION-BASED BOTNET DETECTION

We have introduced the botnet problem, and explained why previous work cannot sufficiently counter this current largest security threat. In this chapter, we introduce our dialog (vertical) correlation-based detection system, BotHunter.

New Approach: We introduce an "evidence-trail" approach to recognizing successful bot infections through the communication sequences that occur during the infection process. We refer to this approach as the infection *dialog correlation* strategy. In dialog correlation, bot infections are modeled as a set of loosely ordered communication flows exchanged between an internal host and one or more external entities. Specifically, we model all bots as sharing a common set of underlying actions that occur during the infection life cycle: target scanning, infection exploit, binary egg download and execution, command and control channel establishment, and outbound attack/propagation. We assume neither that all these events *are required* by all bots nor that every event *will be detected* by our sensor suite. Instead, our dialog correlation system collects an evidence trail of relevant infection events per internal host, looking for a threshold combination of sequences that will satisfy our requirements for bot declaration.

New System: To demonstrate our methodology, we introduce a passive network monitoring system called `BotHunter`, which embodies our infection dialog correlation strategy. The BotHunter correlator is driven by Snort [86] with a customized malware-focused ruleset, which we further augment with two additional bot-specific anomaly-detection plugins for malware analysis: SLADE (Statistical payLoad Anomaly Detection Engine) and SCADE (Statistical sCan Anomaly Detection Engine). SLADE implements a lossy n-gram

payload analysis of incoming traffic, targeting byte-distribution divergences in selected protocols that are indicative of common malware intrusions. SCADE performs several parallel and complementary malware-focused port scan analyses to both incoming and outgoing network traffic. The BotHunter correlator associates inbound scan and intrusion alarms with outbound communication patterns that are highly indicative of successful local host infections. When a sufficient sequence of alerts is found to match BotHunter's infection dialog model, a consolidated report is produced to capture all the relevant events and event participants that contributed to the infection dialog.

Contributions:

- We introduce a new network perimeter monitoring strategy that focuses on detecting malware infections (specifically bots/botnets) through IDS-driven dialog correlation. We present an abstraction of the major network dialog sequences that occur during a successful bot infection, which we call our *bot infection dialog model.*

- Based on this model, we introduce BotHunter, which includes three bot-specific sensors and our IDS-independent dialog correlation engine. BotHunter is the *first* real-time analysis system that can automatically derive a profile of the entire bot detection process, including the identification of the victim, the infection agent, the source of the egg download, and the command and control center. Although our current system implements a classic bot infection dialog model, one can define new models in an XML configuration file and add new detection sensors. Our correlator is IDS-independent, flexible, and extensible to process new models without modification.

- We also present our evaluation of BotHunter against more than 7,000 recent bot infection experiences that we compiled by deploying BotHunter both within a high-interaction honeynet and through a VMware experimentation platform using recently captured bots. We validate our infection sequence model by demonstrating how our correlation engine successfully maps the network traces of a wide variety of recent

bot infections into our model.

Chapter Organization: In Section 3.1, we present our understanding and modeling of the bot infection dialog. In Section 3.2, we detail the design and implementation of BotHunter. In Section 3.3, we describe the evaluation results of BotHunter in several real-world networks. In Section 3.4, we discuss several practical considerations and potential solutions. We introduce the Internet distribution of BotHunter system in Section 3.5 and summarize the chapter in Section 3.6.

3.1 Bot Infection Dialog Model

3.1.1 Understanding Bot Infection Sequences

Understanding the full complexity of the bot infection life cycle is an important challenge for future network perimeter defenses. From the vantage point of the network egress position, distinguishing successful bot infections from the continual stream of background exploit attempts requires an analysis of the two-way dialog flow that occurs between the internal hosts of a network and the Internet. On a well-administered network, the threat of a direct-connect exploit is limited by the extent to which gateway filtering is enabled. However, contemporary malware families are highly versatile in their ability to attack susceptible hosts through email attachments, infected P2P media, and drive-by download infections. Furthermore, with the ubiquity of mobile laptops and virtual private networks (VPNs), direct infection of an internal asset need not necessarily occur across an administered perimeter router. Regardless of how malware enters a host, once established inside the network perimeter, the challenge remains to identify the infected machine and remove it as quickly as possible.

For this study, we focus on a relatively narrow aspect of bot behavior. Our objective is to understand the sequence of network communications and data exchanges that occur between a victim host and other network entities. To illustrate the stages of a bot infection, we outline an infection trace from one example bot, a variant of the Phatbot (aka Gaobot)

family [1]. Figure 4 presents a summary of communication exchanges that were observed during a local host Phatbot infection.

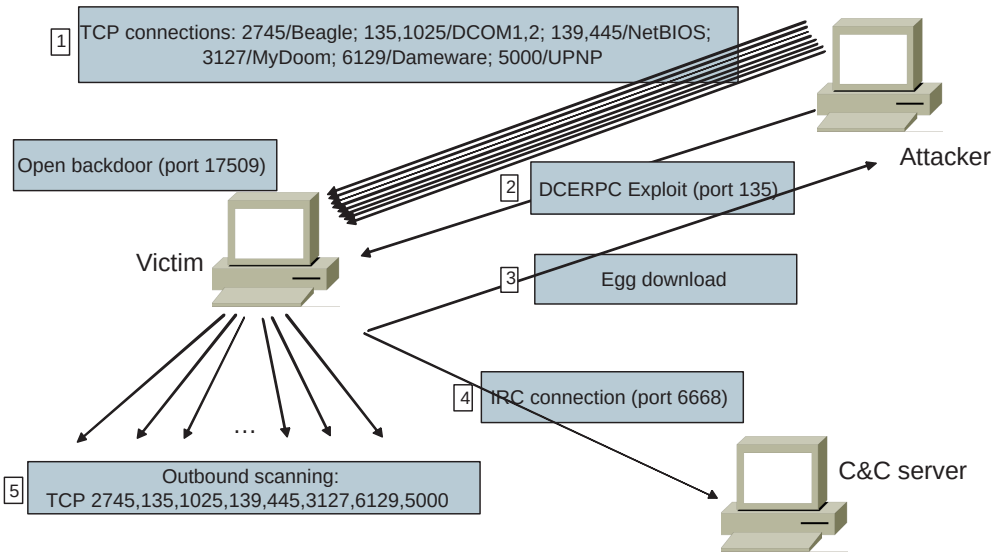


Figure 4: Phatbot infection dialog summary.

As with many common bots that propagate through remote exploit injection, Phatbot first (step 1) probes an address range in search of exploitable network services or responses from Trojan backdoors that may be used to enter and hijack the infected machine. If Phatbot receives a connection reply to one of the targeted ports on a host, it then launches an exploit or logs in to the host using a backdoor. In our experimental case, a Windows workstation replies to a 135-TCP (MS DCE/RPC) connection request, establishing a connection that leads to an immediate RPC buffer overflow (step 2). Once infected, the victim host is directed by an upload shell script to open a communication channel back to the attacker to download the full Phatbot binary (step 3). The bot inserts itself into the system boot process, turns off security software, probes the local network for additional NetBIOS shares, and secures the host from other malware that may be loaded on the machine. The infected victim next distinguishes itself as a bot by establishing a connection to a botnet C&C server, which in the case of Phatbot is established over an IRC channel (step 4). Finally, the newly infected bot establishes a listen port to accept new binary updates and begins scanning other

external victims on behalf of the botnet (step 5).

3.1.2 Modeling the Infection Dialog Process

While Figure 4 presents an example of a specific bot, the events enumerated are highly representative of the life cycle phases that we encounter across the various bot families that we have analyzed. Our bot propagation model is primarily driven by an assessment of outbound communication flows that are indicative of behavior associated with botnet coordination. Whenever possible, we seek to associate such outbound communication patterns with observed inbound intrusion activity. However, this latter activity is not a requirement for bot declaration. Neither are incoming scan and exploit alarms sufficient to declare a successful malware infection, as we assume that a constant stream of scan and exploit signals will be observed from the egress monitor.

We model an infection sequence as a composition of participants and a loosely ordered sequence of exchanges: Infection $I =< A, V, E, C, V', \overline{D} >$, where A = Attacker, V = Victim, E = Egg Download Location, C = C&C Server, and V' = the Victim's next propagation target. $\overline{D}$ represents an infection dialog sequence composed of bidirectional flows that cross the egress boundary. Our infection dialog $\overline{D}$ is composed of a set of five potential dialog transactions (*E1, E2, E3, E4, E5*),[1] some subset of which may be observed during an instance of a local host infection:

– E1: External to Internal Inbound Scan

– E2: External to Internal Inbound Exploit

– E3: Internal to External Binary Acquisition

– E4: Internal to External C&C Communication

– E5: Internal to External Outbound Attack/Propagation

[1]Note, here we have extended the two-phase (preparation and operation) infection proposed in Chapter 2 and Figure 3 into five steps. Here, the preparation phase consists of E1, E2 and E3, and the operation phase consists of the rest two steps, E4 and E5.

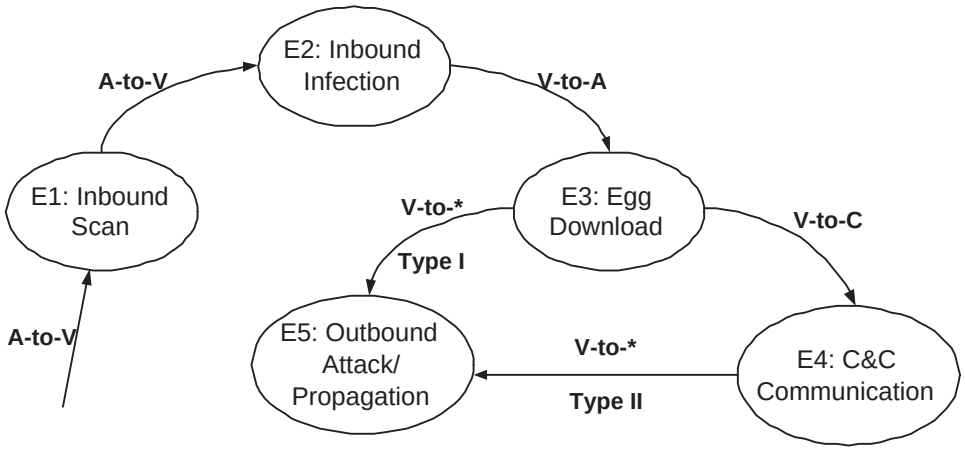


Figure 5: Bot infection dialog model.

Figure 5 illustrates our bot infection dialog model used for assessing bidirectional flows across the network boundary. Our dialog model is similar to the model presented by Rajab et al. in their analysis of 192 IRC bot instances [82]. However, the two models differ in ways that arise because of our specific perspective of egress boundary monitoring. For example, we incorporate early initial scanning, which is often a preceding observation that occurs usually in the form of IP sweeps that target a relatively small set of selected vulnerable ports. We also exclude DNS C&C lookups, which Rajab et al. [82] include as a consistent precursor to C&C coordination, because DNS lookups are often locally handled or made through a designated DNS server via internal packet exchanges that should not be assumed visible from the egress position. Further, we exclude local host modifications because these are also events that are not assumed to be visible from the egress point. Finally, we include internal-to-external attack/propagation, which Rajab et al. [82] exclude. While our model is currently targeted for passive network monitoring events, it will be straightforward to include host-based or DNS-based IDSs that can augment our dialog model.

Figure 5 is not intended to provide a strict ordering of events, but rather to capture a typical infection dialog (exceptions to which we discuss below). In the idealized sequence

of a direct-exploit bot infection dialog, the bot infection begins with an external-to-internal communication flow that may encompass bot scanning (E1) or a direct inbound exploit (E2). When an internal host has been successfully compromised (we observe that many compromise attempts regularly end with process dumps or system freezes), the newly compromised host downloads and instantiates a full malicious binary instance of the bot (E3). Once the full binary instance of the bot is retrieved and executed, our model accommodates two potential dialog paths, which Rajab et al. [82] refer to as the bot Type I versus Type II split. Under Type II bots, the infected host proceeds to C&C server coordination (E4) before attempting attack/propagation. Under a Type I bot, the infected host immediately moves to outbound scanning and attack propagation (E5), representing a classic worm infection.[2]

We assume that bot dialog sequence analysis must be robust to the absence of some dialog events, must allow for multiple contributing candidates for each of the various dialog stages, and must not require strict sequencing on the order in which outbound dialog is conducted. Furthermore, in practice, we have observed that for Type II infections, time delays between the initial infection events (E1 and E2) and subsequent outbound dialog events (E3, E4, and E5) can be significant—sometimes on the order of several hours. Furthermore, our model must be robust to failed E1 and E2 detections, possibly due to insufficient IDS fidelity or due to malware infections that occur through avenues other than direct remote exploit.

One approach to address the challenges of sequence order and event omission is to use a weighted event threshold system that captures the minimum necessary and sufficient sparse sequences of events under which bot profile declarations can be triggered. For example, one can define a weighting and threshold scheme for the appearance of each event such that a minimum set of event combinations is required before bot detection. In our case, we

[2]It is important for BotHunter to capture and report any real worm infection; thus, even though a Type I bot may very well be just a classic worm, it is still included in the model.

assert that bot infection declaration requires a minimum of:

Condition 1: Evidence of a local host infection (E2), AND evidence of outward bot coordination or attack/propagation (E3-E5); or

Condition 2: At least two distinct signs of outward bot coordination or attack/propagation (E3-E5).

In our description of the BotHunter correlation engine in Section 3.2.2, we discuss a weighted event threshold scheme that enforces the above minimum requirement for bot declaration.

3.2 BotHunter: System Design and Implementation

We now turn our attention to the design of a passive monitoring system capable of recognizing the bidirectional warning signs of local host infections, and correlating this evidence against our dialog infection model. Our system, referred to as BotHunter, is composed of a trio of IDS components that monitor inbound and outbound traffic flows, coupled with our dialog correlation engine that produces consolidated pictures of successful bot infections. We envision BotHunter to be located at the boundary of a network, providing it a vantage point to observe the network communication flows that occur between the network's internal hosts and the Internet. Figure 6 illustrates the components within the BotHunter package.

Our IDS detection capabilities are composed on top of the open source release of Snort [86]. We take full advantage of Snort's signature engine, incorporating an extensive set of malware-specific signatures that we developed internally or compiled from the highly active Snort community (e.g., [19] among other sources). The signature engine enables us to produce dialog warnings for inbound exploit usage, egg downloading, and C&C patterns, as discussed in Section 3.2.1.3. In addition, we have developed two custom plugins that complement the Snort signature engine's ability to produce certain dialog warnings.

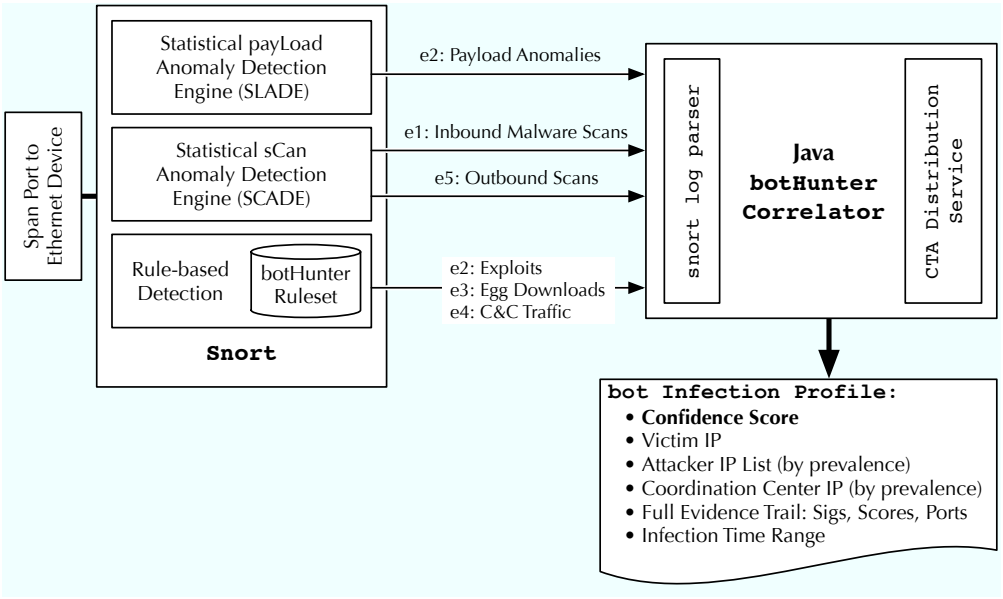


Figure 6: BotHunter system architecture.

We refer to the various IDS alarms as *dialog warnings* because we do not intend the individual alerts to be processed by administrators in search of bot or worm activity. Rather, we use the alerts produced by our sensors as input to drive a bot dialog correlation analysis, the results of which are intended to capture and report the actors and evidence trail of a complete bot infection sequence.

Our two custom BotHunter plugins are called SCADE (Statistical sCan Anomaly Detection Engine) and SLADE (Statistical payLoad Anomaly Detection Engine). SCADE, discussed in Section 3.2.1.1, provides inbound and outbound scan detection warnings that are weighted for sensitivity toward malware-specific scanning patterns. SLADE, discussed in Section 3.2.1.2, conducts a byte-distribution payload anomaly detection of inbound packets, providing a complementary non-signature approach in inbound exploit detection.

The BotHunter correlator, discussed in Section 3.2.2, is responsible for maintaining an assessment of all dialog exchanges, as seen through our sensor dialog warnings, between all local hosts communicating with external entities across the Internet. The BotHunter correlator manages the state of all dialog warnings produced per local host in a data structure we refer to as the *network dialog correlation matrix* (Figure 7). Evidence of local host

infection is evaluated and expired from BotHunter correlator until a sufficient combination of dialog warnings (E1–E5) crosses a weighted threshold. When the bot infection threshold is crossed for a given host, we produce a bot infection profile (illustrated in Figure 11).

Finally, our correlator also incorporates a module that allows users to report bot infection profiles to a remote repository for global collection and evaluation of bot activity. For this purpose, we utilize the Cyber-TA privacy-enabled alert delivery infrastructure [79]. Our delivery infrastructure first anonymizes all source-local addresses reported within the bot infection profile, and then delivers the profile to our data repository through a TLS-over-TOR [35] (onion routing protocol) network connection. These profiles will be made available to the research community, ideally to help in the large-scale assessment of bot dialog behavior, the sources and volume of various bot infections, and for surveying where C&C servers and exploit sources are located.

3.2.1 A Multiple-Sensor Approach to Gathering Infection Evidence

3.2.1.1 SCADE: Statistical sCan Anomaly Detection Engine

Recent measurement studies suggest that modern bots are packaged with around 15 exploit vectors on average [82] to improve opportunities for exploitation. Depending on how the attack source scans its target, we are likely to encounter some failed connection attempts prior to a successful infection.

To address this aspect of malware interaction, we have designed SCADE, a Snort preprocessor plug-in with two modules, one for inbound scan detection (E1 dialog warnings) and another for detecting outbound attack propagations (E5 dialog warnings) once our local system is infected. SCADE E1 alarms provide a potential early bound on the start of an infection, should this scan eventually lead to a successful infection.

Inbound Scan Detection: SCADE is similar in principle to existing scan detection techniques like [57, 86]. However, SCADE has been specifically weighted toward the detection of scans involving the ports often used by malware. It is also less vulnerable to DoS

attacks because its memory trackers do not maintain per-external-source-IP state. Similar to [119], SCADE tracks only scans that are specifically targeted to internal hosts, bounding its memory usage to the number of inside hosts. SCADE also bases its E1 scan detection on failed connection attempts, further narrowing its processing. We define two types of ports: HS (high-severity) ports representing highly vulnerable and commonly exploited services (e.g., 80/HTTP, 135,1025/DCOM, 445/NetBIOS, 5000/UPNP, 3127/MyDoom) and LS (low-severity) ports.[3] Currently, we define 26 TCP and 4 UDP HS ports and mark all others as LS ports. We set different weights to a failed scan attempt to different types of ports. An E1 dialog warning for a local host is produced based on an anomaly score that is calculated as $s = w_1 F_{hs} + w_2 F_{ls}$, where F_{hs} and F_{ls} indicate numbers of cumulative failed attempts at high-severity and low-severity ports, respectively.

Outbound Scan Detection: SCADE's outbound scan detection coverage for E5 dialog warnings is based on a voting scheme (AND, OR or MAJORITY) of three parallel anomaly detection models that track all outbound connections per internal host:

- *Outbound scan rate (s_1):* Detects local hosts that conduct high-rate scans across large sets of external addresses.

- *Outbound connection failure rate (s_2):* Detects abnormally high connection fail rates, with sensitivity to HS port usage. We calculate the anomaly score $s_2 = (w_1 F_{hs} + w_2 F_{ls})/C$, where C is the total number of scans from the host within a time window.

- *Normalized entropy of scan target distribution (s_3):* Calculates a Zipf (power-law) distribution [10] of outbound address connection patterns. A uniformly distributed scan target pattern provides an indication of a potential outbound scan. We use an anomaly scoring technique based on normalized entropy to identify such candidates: $s_3 = \frac{H}{\ln(m)}$, where the entropy of scan target distribution is $H = -\sum_{i=1}^{m} p_i \ln(p_i)$, m

[3]The setting is based on data obtained by analyzing vulnerability reports, malware infection vectors, and analysis reports of datasets collected at `Dshield.org` and other honeynets.

is the total number of scan targets, and p_i is the percentage of the scans at target i.

Each anomaly module issues a sub-alert when $s_i \geq t_i$, where t_i is a threshold. SCADE then uses a user-configurable "voting scheme," i.e., AND, OR, or MAJORITY, to combine the alerts from the three modules. For example, the AND rule dictates that SCADE issues an alert when all three modules issue alerts. The user can choose a proper combination depending on the desired false positive rate (FP).

3.2.1.2 SLADE: Statistical PayLoad Anomaly Detection Engine

SLADE is an anomaly-based engine for payload exploit detection. It examines the payload of every request packet sent to monitored services and outputs an alert if its lossy n-gram frequency deviates from an established normal profile.

SLADE is similar to PAYL [110], which is an anomaly detection scheme based on 1-gram payload byte distribution. PAYL examines the 1-gram byte distribution of the packet payload, i.e., it extracts 256 features each representing the occurrence frequency of one of the 256 possible byte values in the payload. A normal profile for a service/port, e.g., HTTP, is constructed by calculating the average and standard deviation of the feature vector of the normal traffic to the port. PAYL calculates deviation distance of a test payload from the normal profile using a simplified Mahalanobis distance, $d(x, y) = \sum_{i=0}^{255}(|x_i - y_i|)/(\sigma_i + \alpha)$, where y_i is the mean, σ_i is the standard deviation, and α is a smoothing factor. A payload is considered as anomalous if this distance exceeds a predetermined threshold. PAYL is effective in detecting worm exploits with a reasonable false positive rate as shown in [110, 111]. However, it could be evaded by a polymorphic blending attack (PBA) [38]. As discussed in [38, 76, 111], a generic n-gram version of PAYL may help to improve accuracy and the hardness of evasion. The n-gram scheme extracts n-byte sequence information from the payload, which helps in constructing a more precise model of the normal traffic compared to the single-byte (i.e., 1-gram) frequency-based model. In the n-gram scheme the feature space in use is not 256, but 256^n. It is impractical to store and compute in a

256^n dimensional space for a full n-gram scheme when n is large.

SLADE makes the n-gram scheme practical by using a lossy structure while still maintaining approximately the same accuracy as the original full n-gram version. We use a fixed vector counter (with size v) to store a lossy n-gram distribution of the payload. When processing a payload, we sequentially scan n-gram substring str, apply some universal hash function $h()$, and increment the counter at the vector space indexed by $h(str)\ mod\ v$. We then calculate the distribution of the hashed n-gram indices within this (much) smaller vector space v. We define F as the feature space of n-gram PAYL (with a total of 256^n distinct features), and F' as the feature space of SLADE (with v features).

This hash function provides a mapping from F to F' that we utilize for space efficiency. We require only v (e.g., $v = 2,000$), whereas n-gram PAYL needs 256^n (e.g., even for a small n=3, $256^3 = 2^{24} \approx 16M$). The computational complexity in examining each payload is still linear ($O(L)$, where L is the length of payload), and the complexity in calculating distance is $O(v)$ instead of 256^n. Thus, the runtime performance of SLADE is comparable to 1-gram PAYL. Also note that although both using hash techniques, SLADE is different from Anagram [109], which uses a Bloom filter to store all n-gram substrings from normal payloads. The hash function in SLADE is for feature compression and reduction, however the hash functions in Anagram are to reduce the false positives of string lookup in Bloom filter. In essence, Anagram is like a content matching scheme. It builds a huge knowledge base of all known good n-gram substrings using efficient storage and query optimizations provided by bloom filters, and examines a payload to determine whether the number of its n-gram substrings not in the knowledge base exceeds a threshold.

A natural concern of using such a lossy data structure is the issue of accuracy: how many errors (false positives and false negatives) may be introduced because of the lossy representation? To answer this question, we perform the following simple analysis.[4] Let

[4]We consider our analysis not as an exact mathematical proof, but an analytical description about the intuition behind SLADE.

us first overview the reason why the original n-gram PAYL can detect anomalies. We use γ to represent the number of non-zero value features in F for a normal profile used by PAYL. Similarly, γ' is the number of non-zero value features in F' for a normal profile used by SLADE. For a normal payload of $length = L$, there is a total of $l = (L - n + 1)$ n-gram substrings. Among these l substrings, $1 - \beta_n$ percent substrings converge to γ distinct features in the normal profile, i.e., these substrings share similar distributions as the normal profile. The remaining (small portion) β_n percent of substrings are considered as noise substrings that do not belong to the γ features in the normal profile. For a malicious payload, if it can be detected as an anomaly, it should have a much larger portion of noise substrings β_a ($\beta_a > \beta_n$).

We first analyze the false positives when using the lossy structure representation to see how likely SLADE will detect a normal (considered normal by n-gram PAYL) payload as anomalous. For a normal payload, the hashed indices of a $1-\beta_n$ portion of substrings (that converge to γ distinct features in F for the normal profile of PAYL) should now converge in the new vector space (into γ' distinct features in F' for the normal profile of SLADE). Because of the universal hash function, hashed indices of the β_n portion of noise substrings are most likely uniformly distributed into F'. As a result, some of the original noise substrings may actually be hashed to the γ' distinct features in the normal profile of SLADE (i.e., they may not be noise in the new feature space now). Thus, the deviation distance (i.e., the anomaly score) can only decrease in SLADE. Hence, we conclude that SLADE may not have a higher false positive rate than n-gram PAYL.

Now let us analyze the false negative rate, i.e., the likelihood that SLADE will treat a malicious payload (as would be detected by n-gram PAYL) as normal. False negatives happen when the hash collisions in the lossy structure mistakenly map a β_a portion of noise substrings into the γ' features (i.e., the normal profile) for SLADE. By using the universal hash function, the probability for a noise substring to fall into γ' out of v space is $\frac{\gamma'}{v}$. Thus, the probability for all the $l\beta_a$ noise substrings to collide into the γ' portion is about $(\frac{\gamma'}{v})^{l\beta_a}$.

For example, if we assume $v = 2,000, \gamma' = 200, l\beta_a = 100$, then this probability is about $(200/2000)^{100} = 10^{-100} \approx 0$. In practice, the probability of such collisions for partial noise substrings is negligible. Thus, we believe that SLADE does not incur a significant accuracy penalty compared to full n-gram PAYL, while significantly reducing its storage and computation complexity.

We measured the performance of SLADE in comparison to 1-gram PAYL by using the same data set as in [76]. The training and test data sets used were from the first and following four days of HTTP requests from the Georgia Tech campus network, respectively. The attack data consists of 18 HTTP-based buffer overflow attacks, including 11 regular (nonpolymorphic) exploits, 6 mimicry exploits generated by CLET, and 1 polymorphic blending attack used in [38] to evade 2-gram PAYL. In our experiment, we set $n = 4, v = 2,048$.[5]

Table 2 summarizes our experimental results. Here, DFP is the desired false positive rate, i.e., the rejection rate in the training set. RFP is the "real" false positive rate in our test data set. The detection rate is measured on the attack data set and is defined as the number of attack packets classified as anomalous divided by the total number of packets in the attack instances. We conclude from the results that SLADE performs better with respect to both DFP and RFP than the original PAYL (1-gram) system. Furthermore, we discovered that the minimum RFP for which PAYL is able to detect all attacks, including the polymorphic blending attack, is 4.02%. This is usually considered intolerably high for network intrusion detection. On the other hand, the minimum RFP required for SLADE to detect all attacks is 0.3601%. As shown in [76], 2-gram PAYL does not detect the polymorphic blending attack even if we are willing to tolerate an RFP as high as 11.25%. This is not surprising given that the polymorphic blending attack we used was specifically tailored to evade 2-gram PAYL. We also find that SLADE is comparable to (or even better

[5]One can also choose a random v to better defeat evasion attacks like PBA. Also one may use multiple different hash functions and vectors for potentially better accuracy and hardness of evasion.

than) a well-constructed ensemble IDS that combines 11 one-class SVM classifiers [76], and detects all the attacks, including the polymorphic blending attack, for an RFP at around 0.49%. SLADE also has the added advantage of more efficient resource utilization, which results in shorter training and execution times when compared to the ensemble IDS.

Table 2: Performance comparison of 1-gram PAYL and SLADE.

	DFP(%)	0.0	0.01	0.1	1.0	2.0	5.0	10.0
PAYL	RFP(%)	0.00022	0.01451	0.15275	0.92694	1.86263	5.69681	11.05049
	Detected Attacks	1	4	17	17	17	18	18
	Detection Rate(%)	0.8	17.5	69.1	72.2	72.2	73.8	78.6
SLADE	RFP(%)	0.0026	0.0189	0.2839	1.9987	3.3335	6.3064	11.0698
	Detected Attacks	3	13	17	18	18	18	18
	Detection Rate(%)	20.6	74.6	92.9	99.2	99.2	99.2	99.2

3.2.1.3 Signature Engine: Bot-Specific Heuristics

Our final sensor contributor is the Snort signature engine. This module plays a significant role in detecting several classes of dialog warnings from our bot infection dialog model. Snort is our second sensor source for direct exploit detection (E2), and our primary source for binary downloading (E3) and C&C communications (E4). We organize the rules selected for BotHunter into four separate rule files, covering 1046 E2 rules, 71 E3 rules, 246 E4 rules, and a small collection of 20 E5 rules, for total of 1383 heuristics. The rules are primarily derived from the Bleeding-Edge [19] and SourceFire's registered free rulesets.

All the rulesets were selected specifically for their relevance to malware identification. Our rule selections are continually tested and reviewed across operational networks and our live honeynet environment. It is typical for our rule-based heuristics to produce less than 300 dialog warnings per 10-day period monitoring an operational border switch space port of approximately 130 operational hosts (SRI Computer Science Laboratory).

Our E2 ruleset focuses on the full spectrum of external to internal exploit injection attacks, and has been tested and augmented with rules derived from experimentation in our medium and high interactive honeynet environment, where we can observe and validate live malware infection attempts. Our E3 rules focus on (malware) executable download

events from external sites to internal networks, covering as many indications of (malicious) binary executable downloads and download acknowledgment events as are in the publicly available Snort rulesets. Our E4 rules cover internally-initiated bot command and control dialogs, and acknowledgment exchanges, with a significant emphasis on IRC and URL-based bot coordination.[6] Also covered are commonly used Trojan backdoor communications, and popular bot commands built by keyword searching across common major bot families and their variants. A small set of E5 rules is also incorporated to detect well-known internal to external attacks and backdoor sweeps, while SCADE provides the more in-depth hunt for general outbound port scanning.

3.2.2 Dialog-based IDS Correlation Engine

The BotHunter correlator tracks the sequences of IDS dialog warnings that occur between each local host and those external entities involved in these dialog exchanges. Dialog warnings are tracked over a temporal window, where each contributes to an overall infection sequence score that is maintained per local host. We introduce a data structure called the *network dialog correlation matrix*, which is managed, pruned, and evaluated by our correlation engine at each dialog warning insertion point. Our correlator employs a weighted threshold scoring function that aggregates the weighted scores of each dialog warning, declaring a local host infected when a minimum combination of dialog transactions occur within our temporal pruning interval.

Figure 7 illustrates the structure of our *network dialog correlation matrix*. Each dynamically-allocated row corresponds to a summary of the ongoing dialog warnings that are raised between an individual local host and other external entities. The BotHunter correlator manages the five classes of dialog warnings presented in Section 3.1 (E1 through E5), and each event cell corresponds to one or more (possibly aggregated) sensor alerts that map into one of these five dialog warning classes. This correlation matrix dynamically grows when

[6] E4 rules are essentially protocol, behavior and payload content signature, instead of a hard-coded known C&C domain list.

new activity involving a local host is detected, and shrinks when the observation window reaches an interval expiration. The memory usage of the correlation matrix is very efficient because the matrix is bounded by the size of total active internal hosts.

Int. Host	Timer	E1 ◷	E2	E3	E4	E5
192.168.12.1	◷	$A_a \ldots A_b$				
192.168.10.45	◕		$A_c \ldots A_d$		$A_e \ldots A_f$	
192.168.10.66	◕		A_g			
192.168.12.46	◕				$A_h \ldots A_i$	$A_j \ldots A_k$
:						
192.168.11.123	◷ ◕	A_l	$A_m \ldots A_n$	A_o		

Figure 7: BotHunter network dialog correlation matrix.

In managing the dialog transaction history we employ an interval-based pruning algorithm to remove old dialog from the matrix. In Figure 7, each dialog may have one or two expiration intervals, corresponding to a *soft prune timer* (the open-faced clocks) and a *hard prune timer* (the filled clocks). The hard prune interval represents a fixed temporal interval over which dialog warnings are allowed to aggregate, and the end of which results in the calculation of our threshold score. The soft prune interval represents a smaller temporal window that allows users to configure tighter pruning interval requirements for high-production dialog warnings (inbound scan warnings are expired more quickly by the soft prune interval), while the others are allowed to accumulate through the hard prune interval. If a dialog warning expires solely because of a soft prune timer, the dialog is summarily discarded for lack of sufficient evidence (an example is row 1 in Figure 7 where only E1 has alarms). However, if a dialog expires because of a hard prune timer, the dialog threshold score is evaluated, leading either to a bot declaration or to the complete removal of the dialog trace should the threshold score be found insufficient.

To declare that a local host is infected, BotHunter must compute a sufficient and minimum threshold of evidence (as defined in Section 3) within its pruning interval. BotHunter employs two potential criteria required for bot declaration: 1) an incoming infection warning (E2) followed by outbound local host coordination or exploit propagation warnings

(E3-E5), or 2) a minimum of at least two forms of outbound bot dialog warnings (E3-E5). To translate these requirements into a scoring algorithm we employ a regression model to estimate dialog warning weights and a threshold value, and then test our values against a corpus of malware infection traces. We define an expectation table of predictor variables that match our conditions and apply a regression model where the estimated regression coefficients are the desired weights shown in Table 3.

Table 3: Estimated regression coefficients as initial weighting.

	Coefficients	Standard Error
E1	0.09375	0.100518632
E2 rulebase	0.28125	0.075984943
E2 SLADE	0.09375	0.075984943
E3	0.34375	0.075984943
E4	0.34375	0.075984943
E5	0.34375	0.075984943

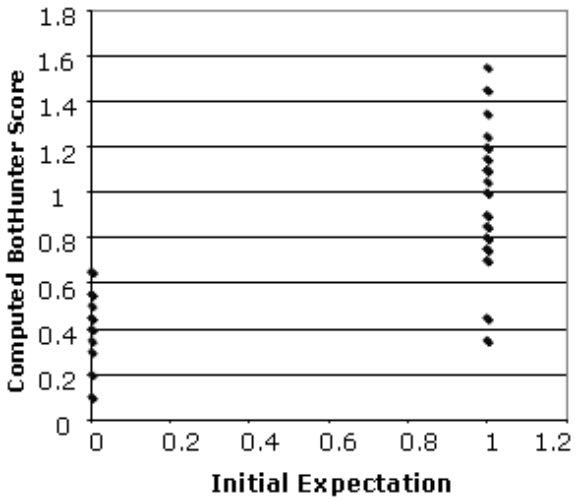


Figure 8: Scoring plot from expectation table.

These coefficients provide an approximate weighting system to match the initial expectation table.[7] We apply these values to our expectation table data to establish a threshold between bot and no-bot declaration. Figure 8 illustrates our results, where bot patterns are

[7]In our model, we define E1 scans and the E2 anomaly score (produced by SLADE) as increasers to infection confidence, such that our model lowers their weight influence.

at X-axis value 1, and non-bot patterns are at X-axis 0. Bot scores are plotted vertically on the Y-axis. We observe that all but one non-bot patterns score below 0.6, and all but 2 bot patterns score above 0.65. Next, we examine our scoring model against a corpus of BotHunter IDS warning sets produced from successful bot and worm infections captured in the SRI honeynet between March and April 2007. Figure 9 plots the actual bot scores produced from these real bot infection traces. All observations produce BotHunter scores of 0.65 or greater.

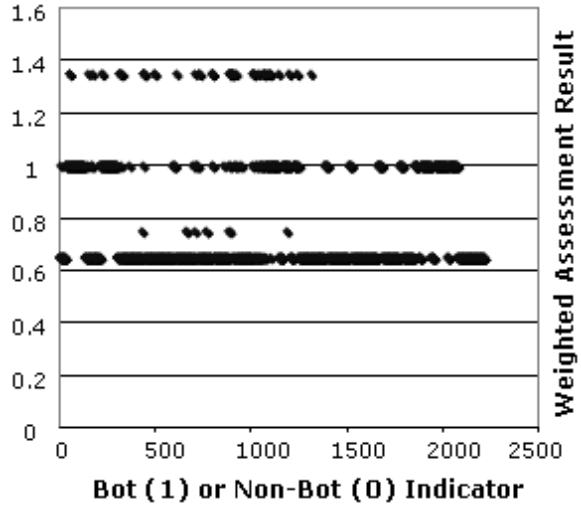


Figure 9: Scoring plot: 2019 real bot infections.

When a dialog sequence is found to cross the threshold for bot declaration, BotHunter produces a *bot profile.* The bot profile represents a full analysis of roles of the dialog participants, summarizes the dialog alarms based on which dialog classes (E1-E5) the alarms map, and computes the infection time interval. Figure 11 provides an example of a bot profile produced by the BotHunter correlation engine. The bot profile begins with an overall dialog anomaly score, followed by the IP address of the infected target (the victim machine), infector list, and possible C&C server. Then it outputs the dialog observation time and reporting time. The raw alerts specific to this dialog are listed in an organized (E1-E5) way and provide some detailed information.

3.3 Evaluating Detection Performance

To evaluate BotHunter's performance, we conducted several controlled experiments as well as real world deployment evaluations. We begin this section with a discussion of our detection performance while exposing BotHunter to infections from a wide variety of bot families using *in situ* virtual network experiments. We then discuss a larger set of true positive and false negative results while deploying BotHunter to a live high-interaction honeynet. This recent experiment exposed BotHunter to 7,204 instances of Windows XP and Windows 2000 direct-exploit malware infections from the Internet. We follow these controlled experiments with a brief discussion of an example detection experience using BotHunter during a live operational deployment.

Next, we discuss our broader testing experiences in two network environments. Here, our focus is on understanding BotHunter's daily false positive (FP) performance, at least in the context of two significantly different operational environments. A false positive in this context refers to the generation of a *bot profile* in response to a non-infection traffic flow, not to the number of IDS dialog warnings produced by the BotHunter sensors. As stated previously, network administrators are not expected to analyze individual IDS alarms. Indeed, we anticipate external entities to regularly probe and attack our networks, producing a regular flow of dialog warnings. Rather, we assert (and validate) that the dialog combinations necessary to cause a bot detection should be rarely encountered in normal operations.

3.3.1 Experiments in an *In situ* Virtual Network

Our evaluation setup uses a virtual network environment of three VMware guest systems. The first is a Linux machine with IRC server installed, which is used as the C&C server, and the other two are Windows 2000 instances. We infect one of the Windows instances and wait for it to connect to our C&C server. Upon connection establishment, we instruct the bot to start scanning and infecting neighboring hosts. We then await the infection and IRC

C&C channel join by the second Windows instance. By monitoring the network activity of the second victim, we capture the full infection dialog. This methodology provides a useful means to measure the false negative performance of BotHunter.

We collected 10 different bot variants from three of the most well-known IRC-based bot families [12]: Agobot/Gaobot/Phatbot, SDBot/RBot/UrBot/UrXBot, and the mIRC-based GTbot. We then ran BotHunter in this virtual network and limited its correlation focus on the victim machine (essentially we assume the HOMENET is the victim's IP). BotHunter successfully detected all bot infections (and produced bot profiles for all).

We summarize our measurement results for this virtual network infection experiment in Table 4. We use Yes or No to indicate whether a certain dialog warning is reported in the final profile. The two numbers within brackets are the number of generated dialog warnings in the whole virtual network and the number involving our victim, respectively. For example, for Phatbot-rls, 2,834 dialog warnings are generated by E2[rb] ([rb] means Snort rule base, [sl] means SLADE), but only 46 are relevant to our bot infection victim. Observe that although many warnings are generated by the sensors, only one bot profile is generated for this infection. This shows that BotHunter can significantly reduce the amount of information a security administrator needs to analyze. In our experiments almost all sensors worked as we expected. We do not see E1 events for RBot because the RBot family does not provide any commands to trigger a vertical scan for all infection vectors (such as the "scan.startall" command provided by the Agobot/Phatbot family). The bot master must indicate a specific infection vector and port for each scan. We set our initial infection vector to DCOM, and since this was successful the attacking host did not attempt further exploits.

Note that two profiles are reported in the gt-with-dcom case. In the first profile, only E2[rb],E2[sl] and E4 are observed. In profile 2, E4 and E5 are observed (which is the case where we miss the initial infection periods). Because this infection scenario is very slow and lasts longer than our 4-minute correlation time window. Furthermore, note that we do

Table 4: BotHunter detection and dialog summary of virtual network infections.

	E1	E2[rb]	E2[sl]	E3	E4	E5
agobot3-priv4	Yes(2/2)	Yes(9/8)	Yes(6/6)	Yes(5)	Yes(38/8)	Yes(4/1)
phat-alpha5	Yes(14/4)	Yes(5,785/5,721)	Yes(6/2)	Yes(3/3)	Yes(28/26)	Yes(4/2)
phatbot-rls	Yes(11/3)	Yes(2,834/46)	Yes(6/2)	Yes(8/8)	Yes(69/20)	Yes(6/2)
rbot0.6.6	No(0)	Yes(2/1)	Yes(2/1)	Yes(2/2)	Yes(65/24)	Yes(2/1)
rxbot7.5	No(0)	Yes(2/2)	Yes(2/2)	Yes(2/2)	Yes(70/27)	Yes(2/1)
rx-asn-2-re-workedv2	No(0)	Yes(4/3)	Yes(3/2)	Yes(2/2)	Yes(59/18)	Yes(2/1)
Rxbot-ak-0.7-Modded.by.Uncanny	No(0)	Yes(3/2)	Yes(3/2)	Yes(2/2)	Yes(73/26)	Yes(2/1)
sxtbot6.5	No(0)	Yes(3/2)	Yes(3/2)	Yes(2/2)	Yes(65/24)	Yes(2/1)
Urx-Special-Ed-UltrA-2005	No(0)	Yes(3/2)	Yes(3/2)	Yes(2/2)	Yes(68/22)	Yes(2/1)
gt-with-dcom-profile1	No(1/0)	Yes(5/3)	Yes(6/2)	No(0)	Yes(221/1)	No(4/0)
gt-with-dcom-profile2	No(1/0)	No(5/0)	No(6/0)	No(0)	Yes(221/44)	Yes(4/2)
gt-with-dcom-10min-profile	No(1/0)	Yes(5/3)	Yes(6/3)	No(0)	Yes(221/51)	Yes(4/2)

not have any detected E3 dialog warnings reported for this infection sequence. Regardless, BotHunter successfully generates an infection profile. This demonstrates the utility of BotHunter's evidence-trail-based dialog correlation model. We also reran this experiment with a 10-minute correlation time window, upon which BotHunter also reported a single infection profile.

3.3.2 SRI Honeynet Experiments

Our experimental honeynet framework has three integral components.

- The first component, *Drone manager*, is a software management component that is responsible for keeping track of drone availability and forwarding packets to various VMware instances. The address of one of the interfaces of this Intel Xeon 3 GHz dual core system is set to be the static route for the unused /17 network. The other interface is used for communicating with the high-interaction honeynet. Packet forwarding is accomplished using network address translation. One important requirements for this system is to keep track of infected drone systems and to recycle uninfected systems. Upon detecting a probable infection (outbound connections), we mark the drone as "tainted" to avoid reassigning that host to another source. Tainted drones are saved for manual analysis or automatically reverted back to previous clean snapshots after a fixed timeout. One of the interesting observations during our study was that most infection attempts did not succeed even on completely unpatched Windows 2000 and

Windows XP systems. As a result, a surprisingly small number of VM instances was sufficient to monitor the sources contacting the entire /17 network.

- The second component is the *high-interaction-honeynet* system, which is hosted in a high-performance Intel Xeon 3 GHz dual core, dual CPU system with 8 GB of memory. For the experiments listed in this chapter, we typically ran the system with 9 Win-XP instances, 14 Windows 2000 instances (with two different service pack levels), and 3 Linux FC3 instances. The system was moderately utilized in this load.

- The final component is the *DNS/DHCP server*, which dynamically assigns IP addresses to VMware instances and also answers DNS queries from these hosts.

Over a 3-month period between May and July 2007, we analyzed a total of 7,204 successful WinXP and Win2K remote-exploit bot infections. Each malware infection instance succeeded in causing the honeypot to initiate outbound communications related to the infection. Through our analysis of these traces using BotHunter sensor logs, we were able to very reliably observe the malware communications associated with the remote-to-local network service infection and the malware binary acquisition (egg download). In many instances we also observed the infected honeypot proceed to establish C&C communications and attempt to propagate to other victims in our honeynet. Through some of this experiment, our DNS service operated unreliably and some C&C coordination events were not observed due to DNS lookup failures.

Figure 10 illustrates a sample infection that was detected using the SRI honeynet, and Figure 11 shows its corresponding BotHunter profile. W32/IRCBot-TO is a recent (released January 19, 2007) bot that propagates through open network shares and affects both Windows 2000 and Windows XP systems [92]. The bot uses the IPC share to connect to the SRVSVC pipe and leverages the MS06-40 exploit [69], which is a buffer overflow that enables attackers to craft RPC requests that can execute arbitrary code. This mechanism is used to force the victim to fetch and execute a binary named netadp.exe from the system

```
 6 <-> <infector-ip> TCP 2971 - <honey-ip> 445 [SYN, SYN,ACK]
13  -> SMB Negotiate Protocol Request
14  <- SMB Negotiate Protocol Response
17  -> SMB Session Setup AndX Request, NTLMSSP_AUTH, User: \
18  <- SMB Session Setup AndX Response
19  -> SMB Tree Connect AndX Request, Path: \\<honey-ip>\IPC\$
20  <- SMB Tree Connect AndX Response
21  -> SMB NT Create AndX Request, Path: \browser
22  <- SMB NT Create AndX Response, FID: 0x4000
23  -> DCERPC Bind: call_id: 0 UUID: SRVSVC
24  <- SMB Write AndX Response, FID: 0x4000, 72 bytes
25  -> SMB Read AndX Request, FID: 0x4000, 4292 bytes at offset 0
26  <- DCERPC Bind_ack
27  -> SRVSVC NetrpPathCanonicalize request
28  <- SMB Write AndX Response, FID: 0x4000, 1152 bytes
29  -> SMB Read AndX Request, FID: 0x4000, 4292 bytes at offset 0

Initiating Egg download
30  <-> <honey-ip> TCP 1028 - <infector-ip> 8295 [SYN, SYNACK]
34-170 114572 byte egg download ...

Connecting to IRC server on port 8080
174 <-> <honey-ip> TCP 1030 - 66.25.XXX.XXX 8080 [SYN, SYNACK]
176 <- NICK [2K|USA|P|00|eOpOgkIc]\r\nUSER 2K-USA
177 -> :server016.z3nnet.net NOTICE AUTH
        :*** Looking up your hostname...\r\n'' ...
179 -> ... PING :B203CFB7
180 <- PONG :B203CFB7
182 -> Welcome to the z3net IRC network ...

Joining channels and setting mode to hidden
183 -> MODE [2K|USA|P|00|eOpOgkIc] +x\r\nJOIN ##RWN irt3hrwn\r\n

Start scanning 203.0.0.0/8
185 -> ....scan.stop -s; .scan.start NETAPI 40 -b -s;
       .scan.start NETAPI 203.x.x.x 20 -s;
       .scan.start NETAPI 20 -a -s;.scan.start SYM 40 -b -s;
       .scan.start MSSQL 40 -b -s\r\n...
191 -> 203.7.223.231 TCP 1072 > 139 [SYN]
192 -> 203.199.174.117 TCP 1073 > 139 [SYN] scan,scan...
```

Figure 10: Honeynet interaction summary for W32/IRCBot-TO.

folder. The infected system then connects to the z3net IRC network and joins two channels upon which it is instructed to initiate scans of 203.0.0.0/8 network on several ports. Other bot families successfully detected by BotHunter included variants of W32/Korgo, W32/Virut.A and W32/Padobot.

Overall, BotHunter detected a total of 7,190 of these 7,204 successful bot infections. This represents a **99.8%** true positive rate. Most malware instances observed during this period transmitted their exploits through ports TCP-445 or TCP-139. This is a very common behavior, as the malware we observe tends to exploit the first vulnerable port that replies to a targeted scans, and ports TCP-445 and TCP-139 are usually among the first ports tested.

This experiment produced 14 bot infections that *did not* produce bot profiles, i.e., a **0.002%** false negative rate. To explain these occurrences we manually examined each bot infection trace that eluded BotHunter, using `Tcpdump` and `Ethereal`. The main

```
Score:           1.95 (>= 0.80)
Infected Target: <honey-ip>
Infector List:   <infector-ip>
C & C List:      66.25.XXX.XXX
Observed Start:  XX/XX/2007 23:46:54.56 PST
Gen. Time:       XX/XX/2007 23:47:13.18 PST

INBOUND SCAN    <unobserved>

EXPLOIT
     event=1:2971 tcp E2[rb] NETBIOS SMB-DS IPC\$
     unicode share access 445<-2971 (23:46:54.56 PST)
     ----------------------------------------
     event=1:99913 tcp E2[rb] SHELLCODE x86 0x90
     unicode NOOP 445<-2971 (23:46:54.90 PST)

EXPLOIT (slade)
     event=552:5555002 (15) tcp E2[sl] Slade detected suspicious
     payload exploit with anomaly score 1843.680342.

EGG DOWNLOAD
     event=1:5001683 tcp E3[rb] Windows executable
     sent potential malware egg 1028<-8295 (01:45:56.69 EST)

C&C TRAFFIC
     event=1:2002023 tcp E4[rb] BLEEDING-EDGE TROJAN
     IRC USER command 1030->8080 (23:47:01.23 PST)
     ----------------------------------------
     event=1:2002024 tcp E4[rb] BLEEDING-EDGE TROJAN
     IRC NICK command 1030->8080 (23:47:01.23 PST)
     ----------------------------------------
     event=1:2002025 tcp E4[rb] BLEEDING-EDGE TROJAN
     IRC JOIN command 1030->8080 (23:47:03.79 PST)

OUTBOUND SCAN
     event=1:2001579 tcp E5[rb] BLEEDING-EDGE Behavioral Unusual Port
     139 traffic, Potential Scan or Infection 1089->139 (01:46:06 EST)
     ----------------------------------------
     event=555:5555005 tcp E5[sc] scade detected scanning of 21 IPs
     (fail ratio=0:0/21): 0->0 (01:46:06 EST)
```

Figure 11: Corresponding BotHunter profile for W32/IRCBot-TO.

reason for these failed bot detections is simply because of infection failures, i.e., the exploit apparently led to instability and eventual failure in the infected host. More commonly, we observed cases in which the infected victim attempted to download the egg or "phone home," but the connection request received no actual reply (most likely the remote host such as C&C server was down).

Lessons: In an earlier evaluation, we had a higher false negative rate. For example, previously in a 3-week period between March and April 2007, BotHunter missed 99 out of 2,019 successful infections, i.e., a 4.9% false negative rate. However, when we investigated the reasons, we found that they were not due to the detection capability of BotHunter. In addition to infection failures in bots as discussed before, we identified some honeynet implementation, setup and policy failures. We observed that our NAT mechanism did not correctly translate application-level address requests (e.g., ftp PORT commands). This prevented several FTP egg download connection requests from proceeding, which would

have otherwise led to egg download detections. In addition, some traces were incomplete due to errors in our honeypot recycling logic which interfered with our observation of the infection logic. Some implementation bugs in honeypot and Drone manager also caused corruption in network traffic data. We then fixed these problems and re-evaluated BotHunter, as showed above.

Discussion: In addition to the above false negative experiences, we also recognize that other reasons could potentially prevent BotHunter from detecting infections. A natural extension of the *infection failures* is for a bot to purposely lay dormant once it has infected a host to avoid association of the infection transmission with an outbound egg download or coordination event. This strategy could be used successfully to circumvent BotHunter deployed with our default fixed pruning interval. While we found some infected victims failed to phone home, we could also envision the egg download source eventually responding to these requests after the BotHunter pruning interval, causing a similar missed association. Sensor coverage is of course another fundamental concern for any detection mechanism. Finally, while these results are highly encouraging, the honeynet environment provided a low-diversity in bot infections, in which attention was centered on direct exploits of TCP-445 and TCP-139. We did not provide a diversity of honeypots with various OSs, vulnerable services, or Trojan backdoors enabled, to fully examine the behavioral complexities of bots.

3.3.3 An Example Detection in a Live Deployment

In addition to our laboratory and honeynet experiences, we have also fielded BotHunter to networks within the Georgia Tech campus network and within an SRI laboratory network. In the next sections we will discuss these deployments and our efforts to evaluate the false positive performance of BotHunter. First, we will briefly describe one example host infection that was detected using BotHunter within our Georgia Tech campus network experiments.

In early February 2007, BotHunter detected a bot infection that produced E1, E4 and E5 dialog warnings. Upon inspection of the bot profile, we observed that the bot-infected machine was scanned, joined an IRC channel, and began scanning other machines during the BotHunter time window. One unusual element in this experience was the omission of the actual infection transmission event (E2), which is observed with high-frequency in our live honeynet testing environment. We assert that the bot profile represents an actual infection because during our examination of this infection report, we discovered that the target of the E4 (C&C Server) dialog warning was an address that was blacklisted both by the ShadowServer (`http://www.shadowserver.org/`) and the botnet mailing list as a known C&C server during the time of our bot profile.

3.3.4 Experiments in a University Campus Network

In this experiment, we evaluate the detection and false positive performance of BotHunter in a production campus network (at the College of Computing, Georgia Tech). The time period of this evaluation was between October 2006 and February 2007.

The monitored link exhibits typical diurnal behavior and a sustained peak traffic of over 100 Mbps during the day. While we were concerned that such traffic rates might overload typical NIDS rulesets and real-time detection systems, our experience shows that it is possible to run BotHunter live under such high traffic rates using commodity PCs. Our BotHunter instance runs on a Linux server with an Intel Xeon 3.6 GHz CPU and 6 GB of memory. The system runs with average CPU and memory utilization of 28% and 3%, respectively.

To evaluate the representativeness of this traffic, we randomly sampled packets for analysis (about 40 minutes). The packets in our sample, which were almost evenly distributed between TCP and UDP, demonstrated wide diversity in protocols, including popular protocols such as HTTP, SMTP, POP, FTP, SSH, DNS, and SNMP, and collaborative applications such as IM (e.g., ICQ, AIM), P2P (e.g., Gnutella, Edonkey, bittorrent), and IRC,

which share similarities with infection dialog (e.g., two-way communication). We believe the high volume of background traffic, involving large numbers of hosts and a diverse application mix, offers an appealing environment to confirm our detection performance, and to examine the false positive question.

First, we evaluated the detection performance of BotHunter in the presence of significant background traffic. We injected bot traffic captured in the virtual network (from the experiments described in Section 3.3.1) into the captured Georgia Tech network traffic. Our motivation was to simulate real network infections for which we have the ground truth information. In these experiments, BotHunter correctly detected all 10 injected infections (by the 10 bots described in Section 3.3.1).

Table 5: Dialog warnings (raw alerts) of BotHunter in 4-month operation in CoC network.

Event	**E1**	**E2[rb]**	**E2[sl]**	**E3**	**E4**	**E5**
Alert#	550,373	950,112	316,467	1,013	697,374	48,063

Next, we conducted a longer-term (4 months) evaluation of false alarm production. Table 5 summarizes the number of dialog warnings generated by BotHunter for each event type from October 2006 to early February 2007. BotHunter sensors generated about 2,563,402 (more than 20,000 per day) raw dialog warnings from all the five event categories. For example, many E3 dialog warnings report on Windows executable downloads, which by themselves do not shed light on the presence of exploitable vulnerabilities. However, our experiments do demonstrate that the alignment of the bot detection conditions outline in Section 3.1 rarely align within a stream of dialog warnings from normal traffic patterns. In fact, only 98 profiles were generated in 4 months, less than one per day on average.

In further analyzing these 98 profiles, we had the following findings. First, there are no false positives related to any normal usage of collaborative applications such as P2P, IM, or IRC. Almost two-thirds (60) of the bot profiles involved access to an MS-Exchange SMB server (33) and SMTP server (27). In the former case, the bot profiles described a NETBIOS SMB-DS IPC$ unicode share access followed by a windows executable downloading

event. Bleeding Edge Snort's IRC rules are sensitive to some IRC commands (e.g., USER) that frequently appear in the SMTP header. These issues could easily be mitigated by additional whitelisting of certain alerts on these servers. The remaining profiles contained mainly two event types and with low overall confidence scores. Additional analysis of these incidents was complicated by the lack of full packet traces in our high-speed network. We can conservatively assume that they are false positives and thereby our experiments here provide a reasonable estimate of the upper bound on the number of false alarms (less than one) in a busy campus network.

3.3.5 Experiments in an Institutional Laboratory

We deployed BotHunter live on a small well-administered production network (a lightly used /17 network that we can say with high confidence is infection free). Here, we describe our results from running BotHunter in this environment. Our motivation for conducting this experiment was to obtain experience with false positive production in an operational environment, where we could also track all network traces and fully evaluate the conditions that may cause the production of any unexpected bot profiles.

BotHunter conducted a 10-day data stream monitoring test from the span port position of an egress border switch. The network consists of roughly 130 active IP addresses, an 85% Linux-based host population, and an active user base of approximately 54 people. During this period, 182 million packets were analyzed, consisting of 152 million TCP packets (83.5%), 15.8 million UDP packets (8.7%), and 14.1 million ICMP packets (7.7%). Our BotHunter sensors produced 5,501 dialog warnings, composed of 1,378 E1 scan events, 20 E2 exploit signature events, 193 E3 egg-download signature events, 7 E4 C&C signature events and 3,904 E5 scan events. From these dialog warnings, the BotHunter correlator produced just one bot profile. Our subsequent analysis of the packets that caused the bot profile found that this was a false alarm. Upon packet inspection, it was found that the session for which the bot declaration occurred consisted of a 1.6 GB multifile FTP transfer,

during which a binary image was transferred with content that matched one of our buffer overflow detection patterns. The buffer overflow false alarm was coupled with a second MS Windows binary download, which caused BotHunter to cross our detection threshold and declare a bot infection.

3.4 Discussion

Several important practical considerations present challenges in extending and adapting BotHunter for arbitrary networks in the future.

Extending BotHunter to Other NIDS: The correlation engine for BotHunter is completely oblivious to Snort's internal structures, intermediate representations, or alert format specifics. The only input to BotHunter is metadata for classification of alerts into their respective E1-E5 categories. Developing similar plug-ins for other NIDS such as Bro [74] is straightforward. In fact, we assume that critical infrastructures would necessarily run multiple sensors to reduce false negatives. BotHunter will seamlessly merge and correlate events from these diverse alert streams to produce a unified bot detector.

Adapting to Emerging Threats and Adversaries: Network defense is a perennial arms race[8] and we anticipate that the threat landscape could evolve in several ways to evade BotHunter. First, bots could use encrypted communication channels for C&C. Second, they could adopt more stealthy scanning techniques. However, the fact remains that hundreds of thousands of systems remain unprotected, attacks still happen in the clear, and adversaries have not been forced to innovate. Detection systems such as BotHunter would raise the bar for successful infections. Moreover, BotHunter could be extended with anomaly-based "entropy detectors" for identification of encrypted channels. We have preliminary results that are promising and defer deeper investigation to future work. We also plan to develop new anomaly-based C&C detection schemes (for E4).

It is also conceivable that if BotHunter is widely deployed, adversaries would devise

[8] In this race, we consider BotHunter to be a substantial technological escalation for the white hats.

clever means to evade the system, e.g., by using attacks on BotHunter's dialog history timers. One countermeasure is to incorporate an additional random delay to the hard prune interval, thereby introducing uncertainty into how long BotHunter maintains local dialog histories.

Incorporating Additional State Logic: The current set of states in the bot infection model was based on the behavior of contemporary bots. As bots evolve, it is conceivable that this set of states would have to be extended or otherwise modified to reflect the current threat landscape. This could be accomplished with simple configuration changes to the BotHunter correlator. We expect such changes to be fairly infrequent as they reflect fundamental paradigm shifts in bot behavior.

Nevertheless, here we show an example of our recent extension [77] on BotHunter's infection dialog model to support detecting P2P spam botnets such as Storm worm [44, 52]. This extension is still compatible with the original infection dialog model, which means it does not affect the detection of bots following the original model. We added two new specific types of events into the infection dialog model, i.e., local asset attack preparation and peer coordination.[9]

- E6: attack preparation. This communication stage represents the locally infected victim performing activities that are indicative of preparing for attack propagation. For example, a high number of contacting multiple mail host IP addresses (e.g., directly connecting to external SMTP port, or issuing DNS MX queries) by a non-SMTP server local asset is a potential precursor action for spam distribution.

- E7: peer coordination. A P2P-based bot solicits and receives coordination instructions from a community of peers within the larger botnet. The protocol is used to synchronize bot actions and accept commands from a hidden controller. For example, in Storm, peer coordination occurs via communications that are overlaid on

[9] Actually, it is better to think that we splited the original E5 into three specific events, E5-E7, including the two additional events and outbound attack/propagation.

the eDonkey UDP P2P protocol. The Storm overlay peer communication has some unique aspects that can be identified by some signature [77].

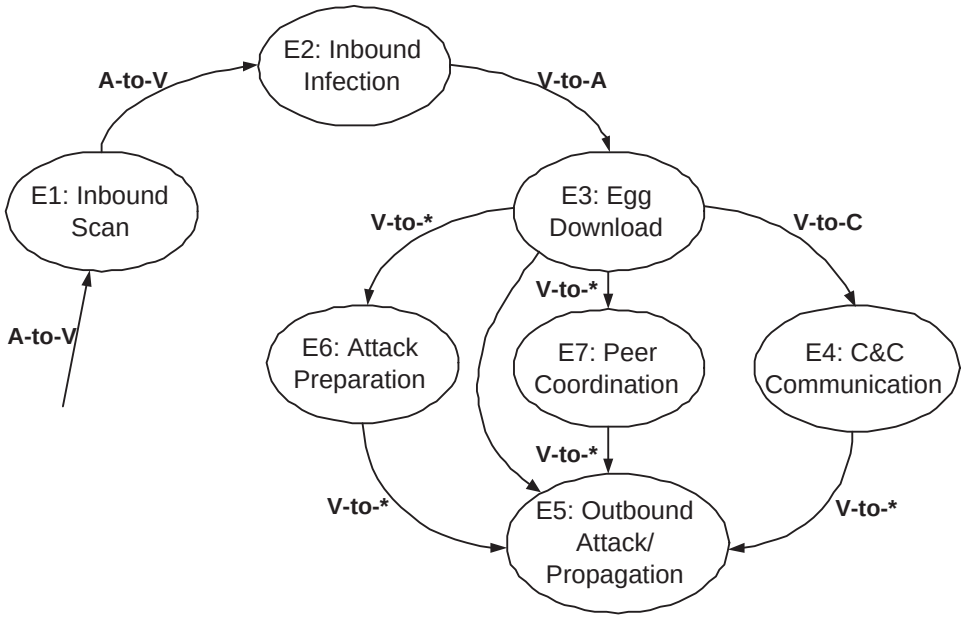


Figure 12: Extended bot infection dialog model.

The extended dialog model is shown in Figure 12. In addition, we added several signature-based rules for E6 and E7, and then simply changed the XML configure file for BotHunter correlator to support the extension. We evaluated the extended BotHunter on real-world captured Storm traces, and successfully detected Storm bots.

3.5 BotHunter Internet Distribution

We are making BotHunter available as a free Internet distribution for use in testing and facilitating research with the hope that this initiative would stimulate community development of extensions.

A key component of the BotHunter distribution is the Java-based correlator that by default reads alert streams from Snort. We have tested our system with Snort 2.6.* and

it can be downloaded from `http://www.cyber-ta.org/botHunter/`.[10] A noteworthy feature of the distribution is integrated support for "large-scale privacy-preserving data sharing." Users can enable an option to deliver secure anonymous bot profiles to the Cyber-TA security repository [79], the collection of which we will make available to providers and researchers. The repository is currently operational and in beta release of its first report delivery software.

Our envisioned access model is similar to that of DShield (`http://www.dshield.org`) with the following important differences. First, our repository is blind to who is submitting the bot report and the system will deliver alerts via TLS over TOR, preventing an association of bot reports to a site via passive sniffing. Second, our anonymization strategy obfuscates all local IP addresses and time intervals in the profile database but preserves C&C, egg download, and attacker addresses that do not match user defined address proximity mask. Users can enable further field anonymizations as they require. We intend to use contributed bot profiles to learn specific alert signature patterns for specific bots, to track attackers, and to identify C&C sites.

3.6 Summary

We have presented the design and implementation of BotHunter, a network perimeter monitoring system for the real-time detection of Internet malware infections. The cornerstone of the BotHunter system is a multi-sensor dialog correlation engine that performs alert consolidation and evidence trail gathering for the investigation of putative infections. We evaluated the system's detection capabilities in an *in situ* virtual network and a live honeynet demonstrating that the system is capable of accurately flagging both well-studied and emergent bots. We also validated low false positive rates by running the system live in two operational production networks. Our experience demonstrates that the system is highly scalable and reliable (very low false positive rates) even with not-so-reliable (weak) raw

[10] In the first five months after its public release, BotHunter has already more than 6,000 downloads.

detectors. BotHunter is also the *first* example of a widely distributed bot infection profile analysis tool. We hope that our Internet release will enable the community to extend and maintain this capability while inspiring new research directions.

CHAPTER IV

BOTSNIFFER: SPATIAL-TEMPORAL CORRELATION-BASED BOTNET DETECTION

We have described our first detection system, BotHunter. It can detect bots that follow an infection model consisting of several infection stages, and it can potentially issue alerts in the early phase of bot infections before bots are fully controlled to perform further malicious activities. However, BotHunter also has some limitations. It is restricted to the *predefined* infection model. In addition, at some stages such as C&C communication, it currently provides only signature-based sensors. In this chapter, we present a new botnet detection system, BotSniffer. This new system does not necessarily require the observation of multiple *different* stages on an individual host, and it does not require botnet-specific signatures.

New Approach: We focus on a new perspective, i.e., horizontal correlation across multiple machines. In particular, since we focus on a specific horizontal correlation that considers both spatial locality (groups of machines) and temporal synchronization (multiple rounds of similar behavior), we refer to this correlation strategy as *spatial-temporal* correlation in the remainder of the chapter. We observe that the bots within a botnet demonstrate spatial-temporal correlations and similarities due to the nature of their pre-programmed response activities to control commands. This invariant helps us identify C&C within network traffic. For instance, at a similar time, the bots within a botnet will execute the same command (e.g., obtain system information, scan the network) and report the progress/result of the task to the C&C server (These reports are likely to be similar in structure and content.) Normal network activities are unlikely to demonstrate such *synchronized* or *correlated* behavior. Using a sequential hypothesis testing algorithm, when we observe multiple

instances of correlated and similar behavior, we can conclude that a botnet is detected with a high probability.

New System: We develop the BotSniffer system to detect mainly *centralized* botnet C&C channels based on spatial-temporal correlation and using network anomaly detection techniques. In particular, we focus on the two commonly used botnet C&C mechanisms, namely, IRC- and HTTP-based C&C channels. For P2P botnet detection (e.g., Nugache [64] and Peacomm [44]), we leave this task to BotMiner, another new system we will describe in the next chapter. BotSniffer monitors two kinds of possible botnet responses, i.e., message response (e.g., IRC `PRIVMSG` message) and activity response (e.g., scan, spam, binary downloading). When there are a group of machines that both share a common IRC or HTTP server connection and demonstrate multiple instances of similar response behavior, it is likely to be a botnet. BotSniffer uses a statistical technique (sequential probability ratio testing) and designs two algorithms (*Response-Crowd-Density-Check* and *Response-Crow-Homogeneity-Check*) for spatial-temporal correlation analysis to detect botnets within a bounded (pre-defined) false positive/negative rate.

Contributions:

- We study two typical styles of control used in centralized botnet C&C. The first is the "push" style, in which commands are pushed or sent to bots. An example of the push style is IRC-based C&C channels. The second is the "pull" style, in which commands are pulled or downloaded by bots. An example of the pull style is HTTP-based C&C channels. Observing the spatial-temporal correlation and similarity nature of these botnet C&Cs, we provide a set of heuristics that distinguish C&C traffic from normal traffic.

- We propose anomaly-based detection algorithms that identify both IRC- and HTTP-based C&Cs in a port-independent manner. Our algorithms include the following advantages: (1) They do not require prior knowledge of C&C servers or content signatures; (2) They are able to detect encrypted C&C; (3) They do not require a

large number of bots to be present in the monitored network, and may even be able to detect a botnet with just a single member in the monitored network in some cases; (4) They have bounded false positive and false negative rates.

- We develop the *BotSniffer* prototype system based on our proposed anomaly detection algorithms. We have evaluated BotSniffer using real-world network traces. The results show that it has high accuracy in detecting real-world botnets using IRC- and HTTP-based C&Cs with a very low false positive rate.

Chapter Organization: In Section 4.1, we provide a background on botnet C&C and the motivation of our botnet detection approach. In Section 4.2, we present the architecture of BotSniffer and describe its detection algorithms in detail. In Section 4.3, we report our evaluation of BotSniffer on various datasets. In Section 4.4, we discuss possible evasions to BotSniffer, the potential corresponding solutions, and future improvement work. We summarize in Section 4.5.

4.1 Botnet C&C and Spatial-Temporal Correlation

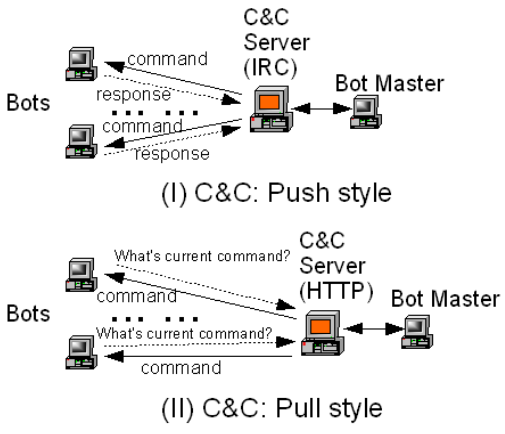
Botnets are different from other forms of malware such as worms in that they use command and control (C&C) channels. It is important to study this botnet characteristic so as to develop effective countermeasures. First, a botnet C&C channel is relatively stable and unlikely to change among bots and their variants. Second, it is the essential mechanism that allows a "botmaster" (who controls the botnet) to direct the actions of bots in a botnet. As such, the C&C channel can be considered the weakest link of a botnet. That is, if we can take down an active C&C or simply interrupt the communication to the C&C, the botmaster will not be able to control his botnet. Moreover, the detection of the C&C channel will reveal both the C&C servers and the bots in a monitored network. Therefore, understanding and detecting the C&C has great value in the battle against botnets.

In this section, we first use case studies to provide a background on two detailed botnet C&C mechanisms, then discuss the invariants of botnet C&C that motivate our detection

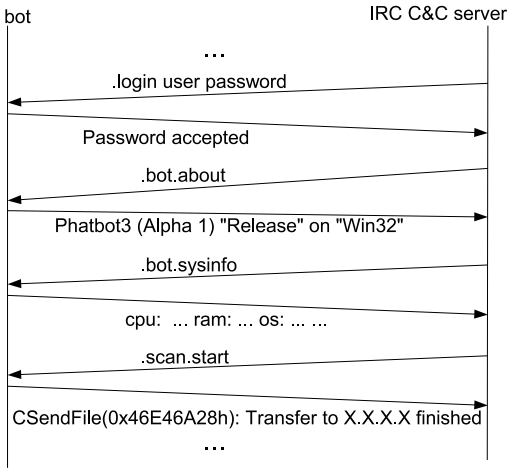
algorithms.

4.1.1 Case Study of Botnet C&C

As shown in Figure 13(a), centralized C&C architecture can be categorized into “push” or “pull” style, depending on how a botmaster’s commands reach the bots.



(a) Two styles of botnet C&C.



(b) An IRC-based C&C communication example.

Figure 13: Centralized botnet command and control: two representative styles and an IRC-based example.

In a push style C&C, the bots are connected to the C&C server (e.g., an IRC server), and wait for commands from the botmaster. The botmaster issues a command in the channel, and all the bots connected to the channel can receive it in real-time. That is, in a push style C&C the botmaster has real-time control over the botnet. IRC-based C&C is the representative example of push style. Many existing botnets use IRC, including the most common bot families such as Phatbot, Spybot, Sdbot, Rbot/Rxbot, GTBot [16]. A botmaster sets up an (or a set of) IRC server(s) as C&C hosts. After a bot is newly infected, it will connect to the C&C server, join a certain IRC channel and wait for commands from the botmaster. Commands will be sent in IRC `PRIVMSG` messages (like a regular chatting message) or a `TOPIC` message. The bots receive commands, understand what the botmaster wants them to do, and execute and then reply with the results. Figure 13(b) shows

a sample command and control session. The botmaster first authenticates himself using a username/password. Once the password is accepted, he can issue commands to obtain some information from the bot. For example, "`.bot.about`" gets some basic bot information such as version, "`.sysinfo`" obtains the system information of the bot-infected machine, and "`.scan.start`" instructs the bots to begin scanning for other vulnerable machines. The bots respond to the commands in pre-programmed fashions. The botmaster has a rich command library to use [16], which enables the botmaster to fully control and utilize the infected machines.

In a pull style C&C, the botmaster simply sets the command in a file at a C&C server (e.g., an HTTP server). The bots frequently connect back to read the command file. This style of command and control is relatively loose in that the botmaster typically does not have real-time control over the bots because there is a delay between the time when he "issues" a command and the time when a bot gets the command. There are several botnets using HTTP protocol for C&C [23, 32, 53, 96]. For example, Bobax [96] is an early HTTP bot designed mainly to send spams. The bots of this botnet periodically connect to the C&C server with a URL such as `http://hostname/reg?u=11111111&v=114`, and receive the command in an HTTP response. The command is in one of the six types, e.g., `prj` (send spams), `scn` (scan others), `upd` (update binary). Botnets can have fairly frequent C&C traffic. For example, in a CERT report [53], researchers report a Web based bot that queries for the command file every 5 seconds and then executes the commands.

Because of its proven effectiveness and efficiency, we expect that centralized C&C (e.g., using IRC or HTTP) will still be widely used by botnets in the near future.

4.1.2 Botnet C&C: Spatial-Temporal Correlation and Similarity

Botnet C&C traffic is known difficult to detect because: (1) it follows normal protocol usage and is similar to normal traffic, (2) the traffic volume is low, (3) there may be very few bots in the monitored network, and (4) may contain encrypted communication.

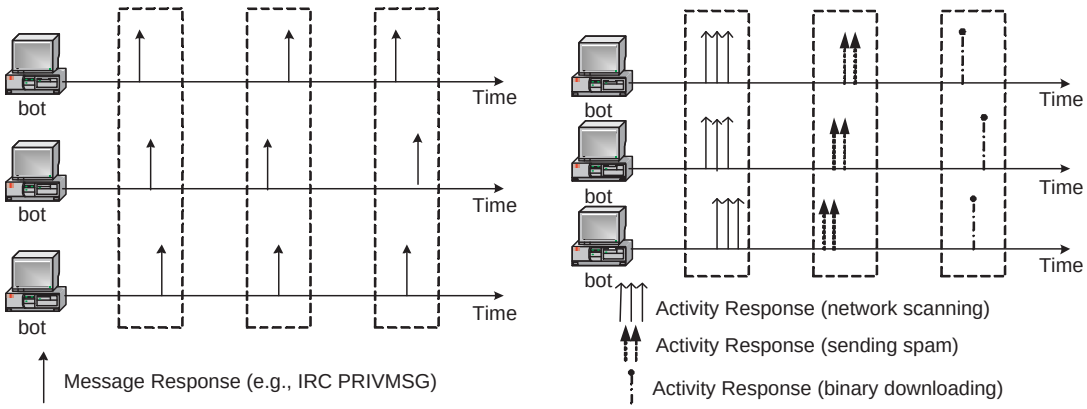
However, we identify several invariants in botnet C&C regardless of the push or pull style. We explain these invariants as follows, which provide the intuition for our detection solution.

First, bots need to connect to C&C servers in order to obtain commands. They may either keep a long connection or frequently connect back. In either case, we can consider that there is a (virtually) long-lived session of C&C channel.[1]

Second, bots need to perform certain tasks and respond to the received commands. We can define two types of responses observable in network traffic, namely, *message* response and *activity* response. A typical example of message response is IRC-based `PRIVMSG` reply as shown in Figure 13(b). When a bot receives a command, it will execute and reply in the same IRC channel with the execution result (or status/progress). The activity responses are the network activities the bots exhibit when they perform the malicious tasks (e.g., scanning, spamming, or binary update) as directed by the botmaster's commands. According to [126], about 53% of botnet commands observed in thousands of real-world IRC-based botnets are related to scan (for spreading or DDoS), about 14.4% are related to binary download (for malware updating). Also, many HTTP-based botnets are mainly used to send spam [96]. Thus, we will observe these malicious activity responses with a high probability [24].

If there are multiple bots in the channel to respond to a command, most of them are likely to respond in a similar fashion. For example, the bots send similar messages or activity traffic at a similar time window, e.g., sending spam as in [83]. Thus, we can observe a *response crowd* of botnet members responding to a command, as shown in Figure 14. Such crowd-like behavior is consistent with all botnet C&C commands and throughout the life-cycle of a botnet. On the other hand, for a normal network service (e.g., an IRC chatting channel), it is unlikely that many clients consistently respond similarly and at a

[1] We consider a session live if the TCP connection is live, or within a certain time window, there is at least one connection to the server.



(a) Message response crowd. (b) Activity response crowd.

Figure 14: Spatial-temporal correlation and similarity in bot responses (message response and activity response).

similar time. That is, the bots have much stronger (and more consistent) synchronization and correlation in their responses than normal (human) users do.

Based on the above observation, our botnet C&C detection approach is aimed at recognizing the spatial-temporal correlation and similarities in bot responses. When monitoring network traffic, as the detection system observes multiple crowd-like behavior, it can declare that the machines in the crowd are bots of a botnet when the accumulated degree of synchronization/correlation (and hence the likelihood of bot traffic) is above a given threshold.

4.2 BotSniffer: Architecture and Algorithms

Figure 15 shows the architecture of BotSniffer. There are two main components, i.e., the monitor engine and the correlation engine. The monitor engine is deployed at the perimeter of a monitored network. It examines network traffic, generates connection record of suspicious C&C protocols, and detects activity response behavior (e.g., scanning, spamming, and binary downloading) and message response behavior (e.g., IRC `PRIVMSG`) in the monitored network. The events observed by the monitor engine are analyzed by the correlation engine. It performs *group analysis* of spatial-temporal correlation and similarity of activity or message response behavior of the clients that connect to the same IRC

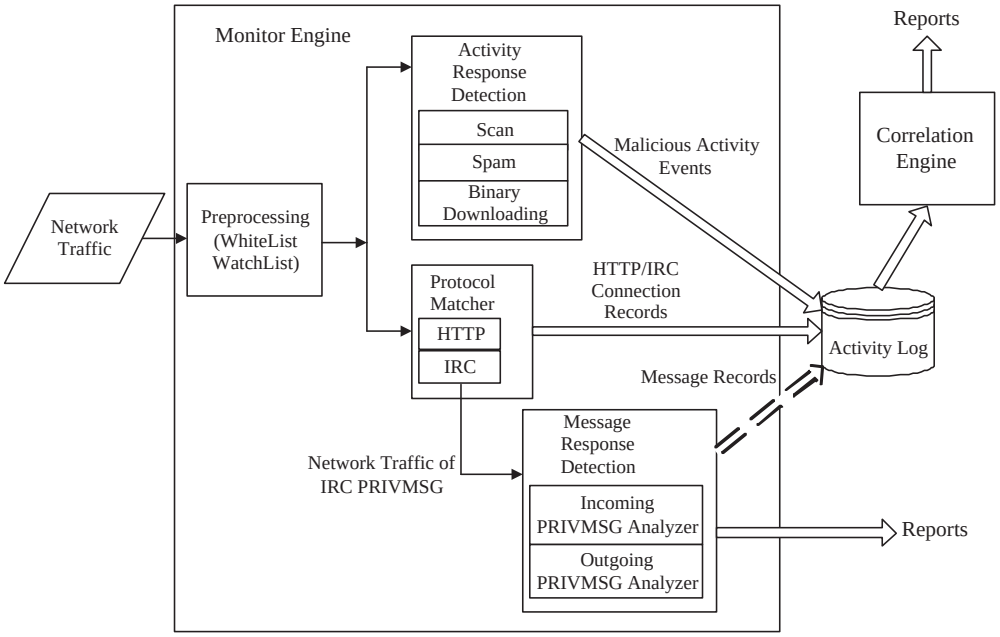


Figure 15: BotSniffer architecture.

or HTTP server. We implemented the monitor engines as several preprocessor plug-ins on top of the open-source system Snort [86], and implemented the correlation engine in Java. We also implemented a real-time *message response* correlation engine (in C), which can be integrated in the monitor engine. The monitor engines can be distributed on several networks, and collect information to a central repository to perform correlation analysis. We describe each BotSniffer component in the following sections.

4.2.1 Monitor Engine

4.2.1.1 Preprocessing

When network traffic enters the BotSniffer monitor engine, BotSniffer first performs preprocessing to filter out irrelevant traffic to reduce the traffic volume. Preprocessing is *not* essential to the detection accuracy of BotSniffer but can improve the efficiency of BotSniffer.

For C&C-like protocol matching, protocols that are unlikely (or at least not yet) used for C&C communications, such as ICMP and UDP, are filtered. We can use a (hard) whitelist to filter out traffic to normal servers (e.g., `Google` and `Yahoo!`) that are less likely to

serve as botnet C&C servers. A soft whitelist is generated for those addresses declared "normal" in the analysis stage, i.e., those clearly declared "not botnet". The difference from a hard list is that a soft list is dynamically generated, and a soft white address is valid only for a certain time window, after which it will be removed from the list.

For activity response detection, BotSniffer can monitor all local hosts or a "watch list" of local clients that are using C&C-like protocols. The watch list is dynamically updated from protocol matchers. The watch list is not required, but if one is available it can improve the efficiency of BotSniffer because its activity response detection component only needs to monitor the network behavior of the local clients on the list.

4.2.1.2 C&C-like Protocol Matcher

We need to keep a record on the clients that are using C&C-like protocols for correlation purpose. Currently, we focus on two most commonly used protocols in botnet C&C, namely, IRC and HTTP. We developed port-independent protocol matchers to find all suspicious IRC and HTTP traffic. This port-independent property is important because many botnet C&Cs may not use the regular ports. We discuss in Section 4.4 the possible extensions.

IRC and HTTP connections are relatively simple to recognize. For example, an IRC session begins with connection registration (defined in RFC1459) that usually has three messages, i.e., `PASS`, `NICK`, and `USER`. We can easily recognize an IRC connection using light-weight payload inspection, e.g., only inspecting the first few bytes of the payload at the beginning of a connection. This is similar to HiPPIE [5]. HTTP protocol is even easier to recognize because the first few bytes of an HTTP request have to be "`GET`," "`POST`," or "`HEAD`."

4.2.1.3 Activity/Message Response Detection

For the clients that are involved in IRC or HTTP communications, BotSniffer monitors their network activities for signs of bot response (message response and activity response).

For message response, BotSniffer monitors the IRC `PRIVMSG` messages for further correlation analysis. For scan activity detection, BotSniffer uses approaches similar to SCADE (Statistical sCan Anomaly Detection Engine) that we have developed for BotHunter [46]. Specifically, BotSniffer mainly uses two anomaly detection modules, namely, the abnormally high scan rate and weighted failed connection rate. BotSniffer uses a new detector for spam behavior detection, focusing on detecting `MX DNS` query (looking for mail servers) and SMTP connections (because normal clients are unlikely to act as SMTP servers). We note that more malicious activity response behavior can be defined and utilized in BotSniffer. For example, binary downloading behavior can be detected using the similar approach as PEHunter [118] and the egg detection method in BotHunter [46].

4.2.2 Correlation Engine

In the correlation stage, BotSniffer first groups the clients according to their destination IP and port pair. That is, clients that connect to the same server will be put into the same group. BotSniffer then performs a *group analysis* of spatial-temporal correlation and similarity. If BotSniffer detects any suspicious C&C, it will issue botnet alerts. In the current implementation, BotSniffer uses the *Response-Crowd-Density-Check* algorithm (discussed in Section 4.2.2.1) for *group activity response* analysis, and the *Response-Crowd-Homogeneity-Check* algorithm (discussed in Section 4.2.2.2) for *group message response* analysis. Any alarm from either of these two algorithms will trigger a botnet alert/report.

BotSniffer also has the ability to detect botnet C&C even when there is only one bot in the monitored network, if certain conditions are satisfied. This is discussed in Section 4.2.3.

4.2.2.1 Response-Crowd-Density-Check Algorithm

The intuition behind this basic algorithm is as follows. For each time window, we check if there is a *dense* response crowd.[2] Recall that a group is a set of clients that connect to the same server. Within this group, we look for any message or activity response behavior. If the fraction of clients with message/activity behavior within the group is larger than a threshold (e.g., 50%), then we say these responding clients form a *dense* response crowd. We use a binary random variable Y_i to denote whether the ith response crowd is dense or not. Let us denote H_1 as the hypothesis "botnet," H_0 as "not botnet." We define $Pr(Y_i|H_1) = \theta_1$ and $Pr(Y_i|H_0) = \theta_0$, i.e., the probability of the ith observed response crowd is dense when the hypothesis "botnet" is true and false, respectively. Clearly, for a botnet, the probability of a dense crowd (θ_1) is high because bots are more synchronized than humans. On the other hand, for a normal (non-botnet) case, this probability (θ_0) is really low. If we observe multiple response crowds, we can have a high confidence that the group is very likely part of a botnet or not part of a botnet.

The next question is how many response crowds are needed in order to make a final decision. To reduce the number of crowds required, we utilize a SPRT (Sequential Probability Ratio Testing [108]) algorithm, which is also known as TRW (Threshold Random Walk [57]), to calculate a comprehensive anomaly score when observing a sequence of crowds. TRW is a powerful tool in statistics and has been used in port scan detection [57] and spam laundering detection [121]. By using this technique, one can reach a decision within a small number of rounds, and with a bounded false positive rate and false negative rate.

TRW is essentially a hypothesis testing technique. That is, we want to calculate the likelihood ratio Λ_n given a sequence of crowds observed $Y_1, ..., Y_n$. Assume the crowds Y_is' are i.i.d. (independent and identically-distributed), we have

[2]We only check when there is at least one client (within the group) that has message/activity response behavior.

$$\Lambda_n = \ln \frac{Pr(Y_1, ..., Y_n|H_1)}{Pr(Y_1, ..., Y_n|H_0)} = \ln \frac{\prod_i Pr(Y_i|H_1)}{\prod_i Pr(Y_i|H_0)} = \sum_i \ln \frac{Pr(Y_i|H_1)}{Pr(Y_i|H_0)}$$

According to the TRW algorithm [57, 108], to calculate this likelihood Λ_n, we are essentially performing a threshold random walk. The walk starts from the origin (0), goes up with step length $ln\frac{\theta_1}{\theta_0}$ when $Y_i = 1$, and goes down with step length $\ln\frac{1-\theta_1}{1-\theta_0}$ when $Y_i = 0$. Let us denote α and β the user-chosen false positive rate and false negative rate, respectively. If the random walk goes up and reaches the threshold $B = \ln\frac{1-\beta}{\alpha}$, this is likely a botnet, and we accept the hypothesis "botnet," output an alert, and stop. If it goes down and hits the threshold $A = \ln\frac{\beta}{1-\alpha}$, it is likely not a botnet. Otherwise, it is pending and we just watch for the next round of crowd.

There are some possible problems that may affect the accuracy of this algorithm.

First, it requires observing multiple rounds of response crowds. If there are only a few responses, the accuracy of the algorithm may suffer. In practice, we find that many common commands will have a long lasting effect on the activities of bots. For example, a single scan command will cause the bots to scan for a long time, and a spam-sending "campaign" can last for a long time [24, 83]. Thus, at least for activity response detection, we can expect to observe sufficient response behavior to have good detection accuracy.

Second, sometimes not all bots in the group will respond within the similar time window, especially when there is a relatively loose C&C. One solution is simply to increase the time window for each round of TRW. Section 4.2.2.2 presents an enhanced algorithm that solves this problem.

To conclude, in practice, we find this basic algorithm works well, especially for *activity* response correlation. To further address the above possible limitations, we next propose an enhanced algorithm.

4.2.2.2 Response-Crowd-Homogeneity-Check Algorithm

The intuition of this algorithm is that, instead of looking at the density of a response crowd, it is important to consider the *homogeneity* of a crowd. A *homogeneous* crowd means that within a crowd, most of the members have very similar responses. For example, the members of a homogeneous crowd have message responses with similar structure and content, or they have scan activities with similar IP address distribution and port range. We note that we currently implement this algorithm only for *message response* analysis. But *activity response* analysis can also utilize this algorithm, as discussed in Section 4.4. In this section, we use *message response* analysis as an example to describe the algorithm.

In this enhanced algorithm, Y_i denotes whether the ith crowd is *homogeneous* or not. We use a clustering technique to obtain the largest cluster of similar messages in the crowd, and calculate the ratio of the size of the cluster over the size of the crowd. If this ratio is greater than a certain threshold, we say $Y_i = 1$; otherwise $Y_i = 0$.

There are several ways to measure the similarity between two messages (strings) for clustering. For example, we can use edit distance (or ED, which is defined as the minimum number of elementary edit operations needed to transform one string into another), longest common subsequence, and DICE coefficient [21]. We require that the similarity metric take into account the structure and context of messages. Thus, we choose DICE coefficient (or DICE distance) [21] as our similarity function. DICE coefficient is based on n-gram analysis, which uses a sliding window of length n to extract substrings from the entire string. For a string X with length l, the number of n-grams is $|ngrams(X)| = l - n + 1$. Dice coefficient is defined as the ratio of the number of n-grams that are shared by two strings over the total number of n-grams in both strings:

$$Dice(X, Y) = \frac{2|ngrams(X) \cap ngrams(Y)|}{|ngrams(X)| + |ngrams(Y)|}$$

We choose $n = 2$ in our system, i.e., we use bi-gram analysis. We also use a simple

variant of hierarchical clustering technique. If there are q clients in the crowd,[3] we compare each of the $\binom{q}{2}$ unique pairs using DICE, and calculate the percentage of DICE distances that are greater than a threshold (i.e., the percentage of similar messages). If this percentage is above a threshold (e.g., 20%), we say the ith crowd is homogeneous, and $Y_i = 1$; otherwise, $Y_i = 0$.

Now we need to set θ_1 and θ_0. These probabilities should vary with the number of clients (q) in the crowd. Thus, we denote them $\theta_1(q)$ and $\theta_0(q)$, or more generally $\theta(q)$. For example, for a homogeneous crowd with 100 clients sending similar messages, its probability of being part of a botnet should be higher than that of a homogeneous crowd of 10 clients. This is because with more clients, it is less likely that by chance they form a homogeneous crowd. Let us denote $p = \theta(2)$ as the basic probability that two messages are similar. Now we have a crowd of q clients, there are $m = \binom{q}{2}$ distinct pairs, the probability of having i similar pairs follows the Binomial distribution, i.e., $Pr(X = i) = \binom{m}{i} p^i (1-p)^{m-i}$. Then the probability of having more than k similar pairs is $Pr(X \geq k) = \sum_{i=k}^{m} \binom{m}{i} p^i (1-p)^{m-i}$. If we pick $k = mt$ where t is the threshold to decide whether a crowd is homogeneous, we obtain the probability $\theta(q) = Pr(X \geq mt)$.

As Figure 16 shows, when there are more than two messages in the crowd and we pick $p \geq 0.6$, the probability $\theta(q)$ is above the diagonal line, indicating that the value is larger than p. This suggests that when we use $\theta_1(2) > 0.6$, we have $\theta_1(q) > \theta_1(2)$. That is, if there are more messages, we will more likely have a higher probability of θ_1. This confirms our intuition that, if it is a botnet, then having more clients (messages) is more likely to form a clustered message group (homogeneous crowd). Also, from the figure, if we pick a small $p \leq 0.3$, we will have $\theta(q) < p$. This suggests that when choosing $\theta_0(2) < 0.3$, we will have much lower probability $\theta_0(q)$ when having multiple messages. Again this confirms the intuition that, for independent users (not a botnet), it is very unlikely for them

[3] Within a certain time window, if a client sends more than one message, the messages will be concatenated together.

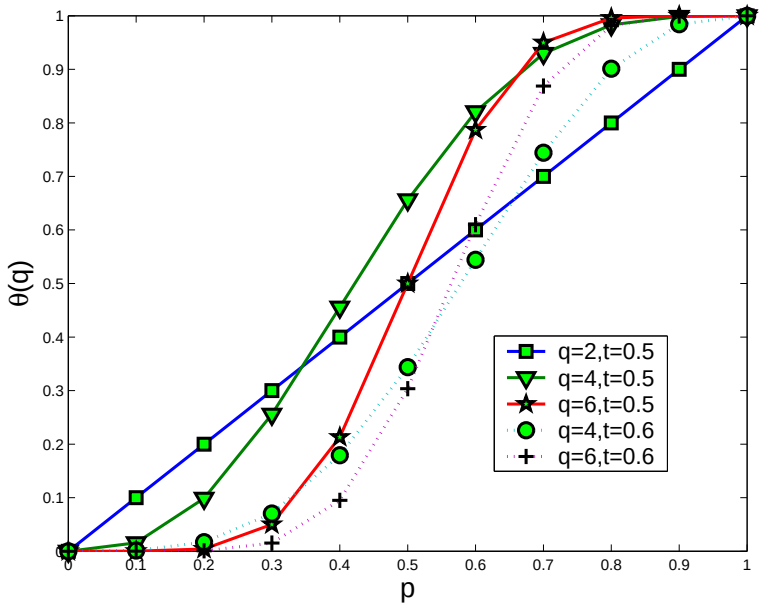


Figure 16: $\theta(q)$, the probability of crowd homogeneity with q responding clients, and threshold t.

to send similar messages. If there are more users, then it is less unlikely they will form a homogeneous crowd because essentially more users will involve more randomness in the messages. In order to avoid calculating $\theta(q)$ all the time, in practice one can pre-compute these probabilities for different q values and store the probabilities in a table for lookup. It may be sufficient to calculate the probabilities for only a few q values (e.g., $q = 3, \ldots, 10$). For $q > 10$, we can conservatively use the probability with $q = 10$.

For the hypothesis "not botnet," for a pair of users, the probability of typing similar messages is very low. To estimate how low this probability is, we first show the probability of typing two similar *length* (size) messages from two chatting users.

Let us use the common assumption of Poisson distribution for the length of messages typed by the user [34] at duration T, $P(X = i) = e^{-\lambda_1 T} \frac{(\lambda_1 T)^i}{i!}$. Then for two independent users, their joint distribution

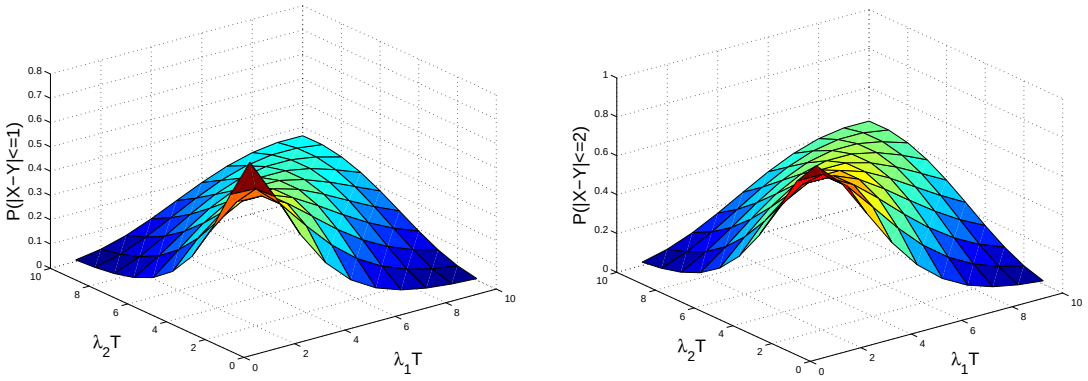
$$P(X = i, Y = j) = P(x = i)P(y = j) = e^{-\lambda_1 T - \lambda_2 T} \frac{(\lambda_1 T)^i (\lambda_1 T)^j}{i!j!}$$

And

$$\begin{aligned} & P(|X-Y| <= \delta) \\ = \; & \textstyle\sum_i P(i,i) + \sum_i P(i,i+1) \\ & + ... + \textstyle\sum_i P(i,i+\delta) \\ & + \textstyle\sum_i P(i,i-1) + ... + \sum_i P(i,i-\delta) \end{aligned} \tag{1}$$

For example,

$$\begin{aligned} & P(|X-Y| <= 1) \\ = \; & \textstyle\sum_i P(i,i) + \sum_i P(i,i+1) + \sum_i P(i,i-1) \\ = \; & e^{-\lambda_1 T - \lambda_2 T} \textstyle\sum_i \frac{(\lambda_1 T)^i (\lambda_2 T)^i}{(i!)^2} (1 + \frac{\lambda_2 T}{i+1} + \frac{i}{\lambda_2 T}) \end{aligned} \tag{2}$$



(a) Probability of $P(|X-Y| <= 1)$ (b) Probability of $P(|X-Y| <= 2)$

Figure 17: Probability of two independent users typing similar length of messages.

Figure 17 illustrates the probability of having two similar length messages from two different users at different settings of λT, the average length of message a user types during T. Figures 17(a) and (b) show the probabilities when two messages have length difference within one character and two characters, respectively. In general, this probability will decrease quickly if the difference between λ_1 and λ_2 increases. Even if two users have the same λ, the probability will also decrease (but slower than the previous case) with the increase of λ. Since two independent users are likely to have different λ values, the probability of typing similar length messages for them is low. For example, if $\lambda_1 T = 5$

and $\lambda_2 T = 10$, the probability $P(|X - Y| <= 2)$ is only around 0.24. If $\lambda_1 T = 5$ and $\lambda_2 T = 20$, this probability will further decrease to 0.0044.

Essentially, if the probability of having two similar *length* messages is low, then the probability of having two similar *content* is even much lower. In correlation analysis, we pick a reasonably conservative value (e.g., 0.15) to estimate the probability of typing similar messages. Even though this value is not precise, e.g., probably higher than the actual value, the only effect is that the TRW algorithm takes a few more rounds to make a decision [57, 108].

In order to make a decision that a crowd is part of a botnet, the expected number of crowd message response rounds we need to observe is:

$$E[N|H_1] = \frac{\beta \ln \frac{\beta}{1-\alpha} + (1-\beta) \ln \frac{1-\beta}{\alpha}}{\theta_1 \ln \frac{\theta_1}{\theta_0} + (1-\theta_1) \ln \frac{1-\theta_1}{1-\theta_0}}$$

where α and β are user-chosen false positive and false negative probabilities, respectively. Similarly, if the crowd is not part of a botnet, the expected number of crowd message response rounds to make a decision is:

$$E[N|H_0] = \frac{(1-\alpha) \ln \frac{\beta}{1-\alpha} + \alpha \ln \frac{1-\beta}{\alpha}}{\theta_0 \ln \frac{\theta_1}{\theta_0} + (1-\theta_0) \ln \frac{1-\theta_1}{1-\theta_0}}$$

These numbers are derived according to [108].

Figure 18 illustrates the expected number of walks ($E[N|H_1]$) (i.e., the number of crowd response rounds need to observe) when the crowd is part of a botnet. Here we fix $\beta = 0.01$ and vary $\theta_0(2)$, $\theta_1(2)$, and α. We can see that even when we have only two clients, and have a conservative setting of $\theta_0(2) = 0.2$ and $\theta_0(2) = 0.7$, it only takes around 6 walks to reach the decision. When we increase $\theta_1(2)$ and decrease $\theta_0(2)$, we can achieve better performance, i.e., fewer rounds of walks. If there are more than two messages (clients), we can have shorter detection time than the case of having only two messages. It is obvious that having more clients in the botnet means that we can make a decision quicker. For example, when $q = 4$, $\theta_1(2) = 0.7$, and $\theta_0(2) < 0.15$, the expected

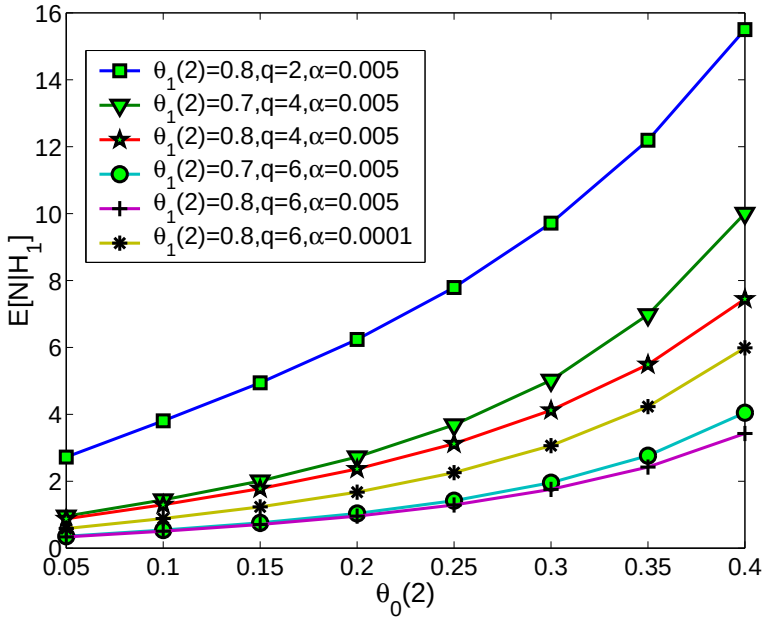


Figure 18: $E[N|H_1]$, the expected number of crowd rounds in case of a botnet (vary $\theta_0(2)$, q, α and fix $\beta = 0.01$).

number of crowd rounds is less than two.

4.2.3 Single Client C&C Detection Under Certain Conditions

Group correlation analysis typically requires having multiple members in a group. In some cases, there is only one client (e.g., the first infected victim) in the group. We recommend a distributed deployment of BotSniffer (as discussed in Section 4.4) to cover a larger network space, and thereby potentially have more clients in a group. Orthogonally, we can use techniques that are effective even if there is only one member in the group, if certain conditions are satisfied.

For IRC communication, a chatting message is usually broadcasted in the channel. That is, every client can see the messages sent from other clients in the same channel (which is the normal operation of IRC chatting service). Thus, every bot should expect to receive the response messages from all other clients. This is essentially similar to the case when we can monitor multiple message responses from multiple clients in the group. We can use the same TRW algorithm here. The only difference is that, instead of estimating

the homogeneity of the outgoing message responses from multiple clients, we estimate the homogeneity of incoming messages (from different users) to a single client. We also implemented BotSniffer to perform this analysis because it complements the algorithms we described in Section 4.2.2.1 and Section 4.2.2.2, especially if there is only one client in the monitored network. Of course, this will not work if the botmaster uses a modified IRC softwares to disable broadcasting messages to every clients in the channel.

For HTTP-based C&C, we notice that bots have strong periodical visiting patterns (to connect back and retrieve commands). *Under this condition*, we can include a new signal encoding and autocorrelation (or self-correlation) approach in BotSniffer to detect such kind of C&C. We describe this technique in the appendix of [48].

Finally, we note that although these two single client detection schemes work well on existing botnet C&C, they are not as robust (evasion-resilient) as the group analysis algorithms discussed in Section 4.2.2.1 and Section 4.2.2.2.

4.3 Experimental Evaluation

To evaluate the performance of BotSniffer, we tested it on several network traces.

4.3.1 Datasets

We have multiple network traces captured from our university campus network. Among those, eight are just port 6667 IRC traffic captured in 2005, 2006, and 2007. Each IRC trace lasts from several days to several months. The total duration of these traces is about 189 days. They were labeled as IRC-n (n = $1, \ldots, 8$). The other five traces are complete packet captures of all network traffic. Two of them were collected in 2004, each lasting about ten minutes. The other three were captured in May and December 2007, each lasting 1 to 5 hours. We labeled them as All-n (n = $1, \ldots, 5$). The primary purpose of using these traces was to test the false positive rate of BotSniffer. We list the basic statistics (e.g., size, duration, number of packets) of these traces in the left part of Table 6.

Table 6: Normal traces statistics (left part) and detection results (right columns) in BotSniffer evaluation.

Trace	trace size	duration	Pkt	TCP flows	(IRC/Web) servers	FP
IRC-1	54MB	171h	189,421	10,530	2,957	0
IRC-2	14MB	433h	33,320	4,061	335	0
IRC-3	516MB	1,626h	2,073,587	4,577	563	6
IRC-4	620MB	673h	4,071,707	24,837	228	3
IRC-5	3MB	30h	19,190	24	17	0
IRC-6	155MB	168h	1,033,318	6,981	85	1
IRC-7	60MB	429h	393,185	717	209	0
IRC-8	707MB	1,010h	2,818,315	28,366	2,454	1
All-1	4.2GB	10m	4,706,803	14,475	1,625	0
All-2	6.2GB	10m	6,769,915	28,359	1,576	0
All-3	7.6GB	1h	16,523,826	331,706	1,717	0
All-4	15GB	1.4h	21,312,841	110,852	2,140	0
All-5	24.5GB	5h	43,625,604	406,112	2,601	0

We also obtained several real-world IRC-based botnet C&C traces from several different sources. One was captured at Georgia Tech honeynet in June 2006. This trace contains about eight hours of traffic (mainly IRC). We labeled it as B-IRC-G. The IRC channel has broadcast on and we can observe the messages sent from other bots in the channel. The trace does not contain the initial traffic, so we did not have the command. From the replies of the clients, it seems like a DDoS attack because bots reported current bandwidth usage and total offered traffic. Besides B-IRC-G, we also obtained two botnet `IRC` logs (not network traces) recorded by an IRC tracker in 2006 [82]. In these logs, there are two distinct IRC servers, so there are two different botnets. We labeled them as B-IRC-J-n (n = 1, 2). In each log, the tracker joined the channel, and sat there watching the messages. Fortunately, the botmaster here did not disable the broadcast, thus, all the messages sent by other bots in the channel were observable.

In addition to these IRC botnet traces, we modified the source codes of three common bots [16] (Rbot, Spybot, Sdbot) and created our version of binaries (so that the bots would only connect to our controlled IRC server). We set up a virtual network environment using VMware and launched the modified bots in several Windows XP/2K virtual machines. We

instructed the bots to connect our controlled C&C server and captured the traces in the virtual network. For Rbot, we used five Windows XP virtual machines to generate the trace. For Spybot and Sdbot, we used four clients. We labeled these three traces as V-Rbot, V-Spybot, and V-Sdbot, respectively. Most of these traces contain both bot message responses and activity responses.

We also implemented two botnets with HTTP-based C&C communication according to the description in [53, 96]. In the first botnet trace, B-HTTP-I, bots regularly connects back to the C&C server every five minutes for commands. We ran four clients in the virtual network to connect to an HTTP server that acted as a C&C server providing commands such as `scan` and `spam`. The four clients are interleaved in time to connect to C&C, i.e., although they periodically connect, the exact time is different because they are infected at different time. In the second trace, B-HTTP-II, we implemented a more stealthy C&C communication. The bot waits a random amount of time for the next connection to the C&C server. This may easily evade the simple autocorrelation-based approach on single client analysis. We wanted to see how it would affect the detection performance of group correlation analysis. These two traces contain bot activity responses.

Table 7 lists some basic statistics of these botnet traces in the left part. Because B-IRC-J-1/2 are not network traces, we report only the number of lines (packets) in the logs.

Table 7: Botnet traces statistics and detection results in BotSniffer evaluation.

BotTrace	trace size	duration	Pkt	TCP flow	Detected
B-IRC-G	950k	8h	4,447	189	Yes
B-IRC-J-1	-	-	143,431	-	Yes
B-IRC-J-2	-	-	262,878	-	Yes
V-Rbot	26MB	1,267s	347,153	103,425	Yes
V-Spybot	15MB	1,931s	180,822	147,921	Yes
V-Sdbot	66KB	533s	474	14	Yes
B-HTTP-I	6MB	3.6h	65,695	237	Yes
B-HTTP-II	37MB	19h	395,990	790	Yes

4.3.2 Experimental Results and Analysis

4.3.2.1 False Positives and Analysis

We first report our experience on the normal traces. We list our detection results in the right part of Table 6. Basically, we list the number of TCP flows (other than TCP flows, we did not count UDP or other flows) and distinct servers (only IRC/Web servers are counted) in the traces. We show the number of IP addresses identified as botnet C&C servers by BotSniffer (i.e., the numbers of false positives) in the rightmost column. Since these traces were collected from well administrated networks, we presumed that there should be no botnet traffic in the traces. We manually verified the raw alerts generated by BotSniffer's monitor engine and also ran BotHunter [46] to confirm that these are clean traces.

The detection results on the IRC traces are very good. Since these traces only contain IRC traffic, we only enabled *message response* correlation analysis engine. On all eight traces (around 189 days' of IRC traffic), BotSniffer only generated a total of 11 FPs on four of the IRC traces. We investigated these alerts and found them all real false positives. There was no false positive (FP) resulted from group analysis. All were generated due to single client incoming message response analysis (Section 4.2.3). The main reason of causing false positives was that, there is still a small probability of receiving very similar messages in a crowd from different users engaging in normal IRC activity. For example, we noticed that in an IRC channel, several users (not in the monitored network) were sending "`@@@@@@@@...`" messages at similar time (and the messages were broadcasted at the channel). This resulted in several homogeneous message response crowds. Thus, our TRW algorithm walked to the hypothesis of "botnet," resulting a FP. While our TRW algorithm cannot guarantee *no* FP, it can provide a pretty good bound of FP. We set $\alpha = 0.005, \beta = 0.01$ in our evaluation and our detection results confirmed the bounds are satisfied because the false positive rate was 0.0016 (i.e., 11 out of 6,848 servers), which is less than $\alpha = 0.005$).

On the network traces All-n, we enabled both *activity response* and *message response*

group analysis engine, and we did not observe false positives. For All-1 and All-2, since the duration is relatively short, we set the time window to one and two minutes, respectively. None of them caused a false positive, because there were very few random scanning activities, which did not cause TRW to make a decision on "botnet." For All-3, All-4 and All-5, we set the time window to 5, 10, and 15 minutes, respectively. Again, we did not observe any false positive. These results showed that our activity response correlation analysis is relatively robust.

4.3.2.2 Detection Accuracy and Analysis

Next, we ran BotSniffer on the botnet traces in two modes, stand alone and mixed with normal traces. It successfully detected all botnet C&C channels in the datasets. That is, it has a detection rate of 100% in our evaluation.

BotSniffer detected B-IRC-G using only message response crowd homogeneity evidences because the trace did not contain activity responses. Since the bots kept sending reports of the attack (which were similar in structure and content) to the C&C server, BotSniffer observed continuous homogeneous message response crowds.

On two IRC logs, we had to adapt our detection algorithms to take a text line as packet. In trace B-IRC-J-1, there were a lot of bots sending similar response messages and these were broadcasted in the IRC channel. BotSniffer easily detected the C&C channel. In trace B-IRC-J-2, although the messages were less often, there were hundred of bots responded almost at the same time, and thus, BotSniffer was able to detect the C&C channels.

On trace V-Rbot, BotSniffer reported botnet alerts because of the group *message response* homogeneity detection and *activity response* (scanning) density detection. Actually, even only one client is monitored in the network, BotSniffer could still detect the botnet C&C because in this case each client could observe messages from other clients in the same botnets. Similarly, BtSniffer also successfully detected C&C channels in traces V-Spybot and V-Sdbot with message responses and/or activity responses.

For traces B-HTTP-I and B-HTTP-II, BotSniffer detected all of the botnets according to activity response group analysis. The randomization of connection periods did not cause a problem as long as there were still several clients performing activity responses at the time window.

4.3.2.3 Summary

In our experiments, BotSniffer successfully detected all botnets and generated very few false positives. In addition, its correlation engine generated accurate and concise report rather than producing alerts of malicious events (e.g., scanning, spamming) as a traditional IDS does. For instance, in trace All-4, the monitor engine produced over 100 activity events, none of which is the indication of actual botnets (e.g., they are false positives), while the correlation engine did not generate a false positive. In another case, e.g., in V-Spybot, there were over 800 scanning activity events produced by the monitor engine, and the correlation engine only generated one botnet report (true positive), which was a great reduction of work for administrators.

In terms of performance comparison with existing botnet detection systems, we can mainly do a paper-and-pencil study here because we could not obtain these tools, except BotHunter [46]. Rishi [43] is a relevant system but it is signature-based (using known knowledge of bot nicknames). Thus, if IRC bots simply change their nickname pattern (for example, many of botnets in our data do not have regular nickname patterns), Rishi will miss them. However, such changes will not affect BotSniffer because it is based on the response behavior. Another relevant work is the BBN system [66, 98]. Its detection approach is based on clustering of some general network-level traffic features (such as duration, bytes per packet). Such approach is easy to evade by simply changing the network flows. It can potentially have more false positives because it does not consider the temporal synchronization and correlation of responses. BotHunter [46] is a bot detection system using IDS-based dialog correlation according to a user-defined bot infection live-cycle model.

It cannot detect bots given only IRC communication. Its current C&C detection module relies on known signatures, and thus, it fails on some botnet traces (e.g., B-IRC-G, B-HTTP-I). The anomaly-based IRC botnet detection system in [18] has the similar problem as BotHunter. Without considering the *group spatial-temporal correlation and similarity*, these systems may also have a higher false positive rate than BotSniffer.

Although BotSniffer performed well in our evaluation, it can fail to detect botnets in several cases. We next discuss these issues and the possible solutions, as well as future work on improving BotSniffer.

4.4 Discussion

As we have stated, BotSniffer can still have false positives if normal hosts happen to behavior very similarly at a similar time, though this probability is very low. In the experiments, we have shown such examples. There might be other examples, e.g., flash crowds. That is, the same set of hosts visit the same website (e.g., `slashdot.org`) and then download the same binary (probably from other websites) at a similar time. If this occurs several rounds, BotSniffer is likely to trigger an alert. However, we envision such cases to be rare and we can use a white list to effectively reduce such false positives.

Next, we discuss more important issues related to false negatives.

4.4.1 Possible Evasions and Solutions

Evasion by misusing the whitelist: If a botmaster knows our hard whitelist, he may attempt to misuse these white addresses. For example, he can use them as third-party proxies for C&C purpose to bypass the detection of BotSniffer. However, as we discussed earlier, a whitelist is not essential to BotSniffer and mainly serves to improve its efficiency. Thus, whitelists can be removed to avoid such evasions. In another evasion case, an adversary controlling the C&C server may attempt to first behave normally and trick BotSniffer to decide that the C&C server is a normal server and put the server address in the soft whitelist. After that, the adversary begins to use the C&C server to command the bots to perform real

malicious activities. To defeat this evasion, for each address being added to soft whitelist, we can keep a *random* and short timer so that the address will be removed when the timer expires. Thus, the adversary's evasion attempt will not succeed consistently.

Evasion by encryption: Botnets may still use known protocols (IRC and HTTP) that BotSniffer can recognize, but the botmasters can encrypt the communication content to attempt to evade detection. First of all, this may *only* mislead *message response* correlation analysis, but *cannot* evade *activity response* correlation analysis. Second, we can improve *message response* correlation analysis to deal with encrypted traffic. For example, instead of using simple DICE distance to calculate the similarity of two messages, we can use information-theoretic metrics that are relatively resilient to encryption, such as entropy, or normalized compression distance (NCD [14, 117]), which is based on Kolmogorov complexity.

Evading protocol matcher: Although botnets tend to use existing common protocols to build their C&C, they may use some obscure protocols or even create their own protocols.[4] It is worth noting that "push" and "pull" are the two representative C&C styles. Even when botnets use other protocols, the spatial-temporal correlation and similarity properties in "push" and "pull" will remain. Thus, our detection algorithms can still be used after new protocol matchers are added. We can develop a generic C&C-like protocol matcher that uses traffic features such as BPP (bytes per packet), BPS (bytes per second), and PPS (packet per second) [66, 98] instead of relying on protocol keywords. This protocol matching approach is based on the observation that there are generic patterns in botnet C&C traffic regardless of the protocol being used. For example, C&C traffic is typically low volume with a just a few packets in a session and a few bytes in a packet. Ultimately, to overcome the limitations of protocol matching and protocol-specific detection techniques,

[4]However, a brand new protocol itself is suspicious already. A botnet could also exploit the implementation vulnerability of protocol matchers. For example, if an IRC matcher only checks the first ten packets in a connection to identify the existence of IRC keywords, the botmaster may have these keywords occur after the first ten packets in order to evade this protocol matcher.

we have developed a next-generation botnet detection system, BotMiner, which is independent of the protocol and network structure used for botnet C&C, as will be described in the next chapter.

Evasion by using very long response delay: A botmaster may command his bots to wait for a very long time (e.g., days or weeks) before performing message or malicious activity response. In order to detect such bots using BotSniffer, we have to correlate IRC or HTTP connection records and activity events within a relatively long time window. In practice, we can perform correlation analysis using multiple time windows (e.g., one hour, one day, one week, etc.). However, we believe that if bots are forced to use a *very long* response delay, the utility of the botnet to botmaster is reduced or limited because the botmaster can no longer command his bots promptly and reliably. For example, the bot-infected machines may be powered off or disconnected from the Internet by the human users/owners during the delay and become unavailable to the botmaster. We can also use the analysis of *activity response crowd homogeneity* (see Section 4.4.2) to defeat this evasion. For example, if we can observe over a relatively long time window that several clients are sending spam messages with *very similar* contents, we may conclude that the clients are part of a botnets.

Evasion by injecting random noise packet, injecting random garbage in the packet, or using random response delay: Injecting random noise packet and/or random garbage in a packet may affect the analysis of *message* response crowd homogeneity. However, it is unlikely to affect the *activity* response crowd analysis as long as the bots still need to perform the required tasks. Using random message/activity response delay may cause problems to the *Response-Crowd-Density-Check* algorithm because there may not be sufficient number of responses seen within a time window for one round of TRW. However, the botmaster may lose the reliability in controlling and coordinating the bots promptly if random response delay is used. We can use a larger time window to capture more responses. Similar to evasion by long response delay discussed above, for evasion by random response

delay, a better solution is to use the analysis of *activity response crowd homogeneity* (see Section 4.4.2).

In summary, although it is not perfect, BotSniffer greatly enhances and complements the capabilities of existing botnet detection approaches. Further research is needed to improve its effectiveness against the more advanced and evasive botnets.

4.4.2 Improvements to BotSniffer

Activity response crowd homogeneity check: We have already discussed homogeneity analysis of *message response crowd* in Section 4.2.2.2. We can perform similar check on the homogeneity of *activity response crowd*. For instance, for scanning activity, we consider two scans to be similar if they have similar distribution or entropy of the target IP addresses and similar ports. A similarity function of two spam activities can be based on the number of common mail servers being used, the number of spam messages being sent, and the similarity of spam structure and content (e.g., the URLs in the messages). A similarity function of two binary downloading activities can be based on the byte value distribution or entropy of the binary or binary string distance. By including *Response-Crowd-Homogeneity-Check* on activity responses, in addition to the similar check on message responses, we can improve the detection accuracy of BotSniffer and its resilience to evasion.

Combine more features in analysis: As with other detection problems, including more features can improve the accuracy of a botnet detection algorithm. For example, we can check whether there are any user-initiated queries, e.g., `WHO`, `WHOIS`, `LIST`, and `NAMES` messages, in an IRC channel. The intuition is that a bot is unlikely to use these commands like a real user. To detect an IRC channel that disables broadcast (as in the more recent botnets), we can consider the message exchange ratio, defined as $\frac{m_i}{m_o}$, i.e., the ratio between the number of incoming `PRIVMSG` messages (m_i) and the number of outgoing `PRIVMSG` messages (m_o). The intuition is that for a normal (broadcasting) IRC

channel, most likely there are multiple users/clients in the chatting channel, and a user usually receives more messages (from all other users) than he sends. On the other hand, in the botnet case with broadcast disabled, the number of incoming messages can be close to the number of outgoing messages because a client cannot see/receive the messages sent by other clients. The number of incoming messages can also be smaller than the number of outgoing messages, for example, when there are several packets/responses from a bot corresponding to one botmaster command, or when the botmaster is not currently online sending commands. In addition, we can consider other group similarity measures on traffic features, e.g., duration, bytes per second, and packets per second.

Distributed deployment on Internet: Ideally, BotSniffer deployment should be scalable, i.e., it should be able to handle a large volume of traffic and cover a large range of network addresses. We envision that BotSniffer can be distributed in that many monitor sensors can be deployed in distributed networks and report to a central repository that also performs correlation and similarity analysis.

4.5 Summary

Botnet detection is a relatively new and a very challenging research area. In this chapter, we presented BotSniffer, a network anomaly based botnet detection system that explores the spatial-temporal correlation and similarity properties of botnet command and control activities. Our detection approach is based on the intuition that since bots of the same botnet run the same bot program, they are likely to respond to the botmaster's commands and conduct attack/fraudulent activities in a similar fashion. BotSniffer employs several correlation and similarity analysis algorithms to examine network traffic and identifies the crowd of hosts that exhibit very strong synchronization/correlation in their responses/activities as bots of the same botnet. We reported an experimental evaluation of BotSniffer on many real-world network traces and showed that it has very promising detection accuracy with a very low false positive rate.

CHAPTER V

BOTMINER: HORIZONTAL CORRELATION-BASED, PROTOCOL- AND STRUCTURE-INDEPENDENT BOTNET DETECTION

Botnets are evolving and quite flexible. We have witnessed that the protocols used for C&C evolved from IRC to others (e.g., HTTP [23, 32, 53, 96]), and the structure moved from centralized to distributed (e.g., P2P [44, 64]). Furthermore, during its lifetime, a botnet can also frequently change its C&C server address, e.g., using fast-flux service networks [51].

Previously, we have described BotHunter and BotSniffer. BotHunter [46] is capable of detecting bots regardless of the C&C structure and network protocol as long as the bot behavior follows a *pre-defined* infection life-cycle dialog model. However, it may fail as soon as botnets change their infection model(s). BotSniffer is designed mainly for detecting botnets using *centralized* C&C protocols such as IRC or HTTP. Similarly, many other existing approaches such as [18, 43, 59, 66, 98, 124] are restricted to detecting botnets using *centralized* C&C, and mostly only for IRC-based botnets. Thus, the aforementioned detection approaches may become ineffective against botnets once they evolve their C&C techniques.

In this chapter, we introduce a new detection system, BotMiner, which is based on a concept of horizontal correlation similar to that of BotSniffer to detect *a group* of compromised machines inside the monitored network that are part of a botnet. However, we propose a more general technique framework that can detect both centralized and *P2P* botnets.

Revisit the Definition of Botnet: To design a general detection approach that can resist the evolution and changes in botnet C&C techniques, we need to study the *intrinsic*

botnet communication and activity characteristics that remain detectable with the proper detection features and algorithms. We thereby revisit the definition of a botnet, i.e., "a *coordinated group* of *malware* instances (bots) that are *controlled* by a botmaster via some command and control (C&C) channel." The term "malware" means these bots are used to perform *malicious activities*. For example, according to [126], about 53% of botnet activity commands observed in thousands of real-world IRC-based botnets are related to scan (for the purpose of spreading or DDoS),[1] and about 14.4% are related to binary downloading (for the purpose of malware updating). In addition, most of HTTP-based and P2P-based botnets are used to send spam [44, 96]. The term "controlled" means these bots have to contact their C&C servers to obtain commands to carry out activities, e.g., to scan. In other words, there should be *communication between bots and C&C servers/peers* (which can be centralized or distributed). Finally, the term "coordinated group" means that multiple (at least two) bots within the same botnet will perform *similar* or *correlated* C&C communications and malicious activities. If the botmaster commands each bot individually with a different command/channel, the bots are nothing but some isolated/unrelated infections. That is, they do not function as a bot*net* according to our definition and are out of the scope of this work.[2]

New Approach and System: We develop the BotMiner detection system based on the above essential properties of botnets. BotMiner monitors both *who is talking to whom* that may suggest C&C communication activities and *who is doing what* that may suggest malicious activities, and finds a *coordinated group pattern* in both kinds of activities. More specifically, BotMiner clusters similar communication activities in the *C-plane* (C&C communication traffic), clusters similar malicious activities in the *A-plane* (activity traffic), and

[1] For spreading, the scans usually span many different hosts (within a subnet) indicated by the botnet command. For DDoS, usually there are numerous connection attempts to a specific host. In both cases, the traffic can be considered as scanning related.

[2] One can use our complementary systems such as BotHunter to detect *individual* bots.

performs cross-cluster correlation to identify the hosts that share both similar communication patterns *and* similar malicious activity patterns. These hosts, according to the botnet definition and properties discussed above, are bots in the monitored network.

Contributions:

- We develop a novel, general, horizontal correlation-based botnet detection framework that is grounded on the definition and essential properties of botnets. Our detection framework is independent of botnet C&C protocol and structure, resistant to changes in the location of the C&C server(s), and requires no *a priori* knowledge (e.g., C&C addresses/signatures) of specific botnets. It can detect both centralized (e.g., IRC,HTTP) and current (and possibly future) P2P-based botnets.

- We define a new "aggregated communication flow" (C-flow) record data structure to store aggregated traffic statistics, and design a new layered clustering scheme with a set of traffic features measured on the C-flow records. Our clustering scheme can accurately and efficiently group similar C&C traffic patterns. The technique is also independent of the content of the C&C communication. That is, we do not inspect the content of the C&C communication, because C&C could be encrypted or use a customized (obscure) protocol.

- We build the BotMiner prototype system based on our general detection framework, and evaluate it with multiple real-world network traces including normal traffic and several real-world botnet traces that contain IRC-, HTTP- and P2P-based botnet traffic (including Nugache and Storm). The results show that BotMiner has a high detection rate and a low false positive rate.

Chapter Organization: In Section 5.1, we describe the problem statement and assumptions of BotMiner. In Section 5.2, we describe the architecture, detection algorithms and implementation. In Section 5.3, we describe our evaluation on various real-world network traces. In Section 5.4, we discuss current limitations and possible solutions. We

conclude the chapter in Section 5.5.

5.1 Problem Statement and Assumptions

In order for a botmaster to command a botnet, there needs to be a C&C channel through which bots receive commands and coordinate attacks and fraudulent activities. The C&C channel is the means by which individual bots form a bot*net*. Centralized C&C structures using the Internet Relay Chat (IRC) protocol have been utilized by botmasters for a long time. In this architecture, each bot logs into an IRC channel, and seeks commands from the botmaster. Even today, many botnets are still designed this way. Quite a few botnets, though, have begun to use other protocols such as HTTP [23, 32, 53, 96], probably because HTTP-based C&C communications are more stealthy given that Web traffic is generally allowed in most networks. Although centralized C&C structures are effective, they suffer from the single-point-of-failure problem. For example, if the IRC channel (or the Web server) is taken down due to detection and response efforts, the botnet loses its C&C structure and becomes a collection of isolated compromised machines. Recently, botmasters began using peer-to-peer (P2P) communication to avoid this weakness. For example, Nugache [64] and Storm worm [44, 52] (a.k.a. Peacomm) are two representative P2P botnets. Storm, in particular, distinguishes itself as having infected a large number of computers on the Internet and effectively becoming one of the "world's top super-computers" [61] for the botmasters.

Figure 19 illustrates the two typical botnet structures, namely *centralized* and *P2P*. The bots receive commands from the botmaster using a *push* or *pull* mechanism [48] and execute the assigned tasks.

The operation of a centralized botnet is relatively easy and intuitive [48], whereas this is not necessarily true for P2P botnets. Therefore, here we briefly illustrate an example of a typical P2P-based botnet, namely Storm worm [44, 52]. In order to issue commands to the bots, the botmaster publishes/shares command files over the P2P network, along with

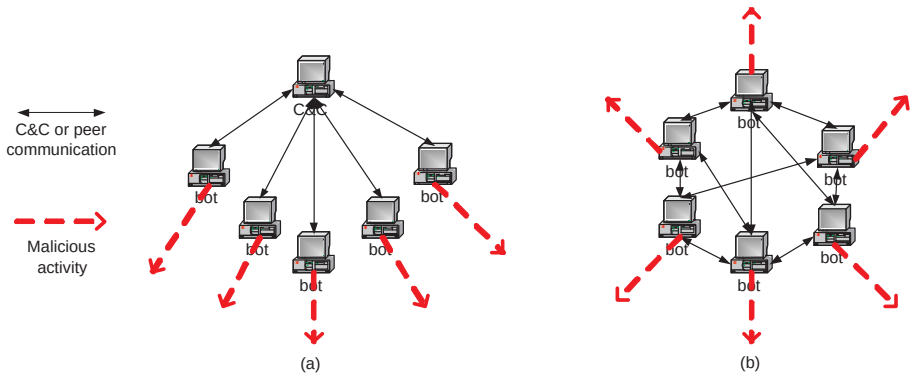


Figure 19: Possible C&C structures of a botnet: (a) centralized; (b) peer-to-peer.

specific search keys that can be used by the bots to find the published command files. Storm bots utilize a pull mechanism to receive the commands. Specifically, each bot frequently contacts its neighbor peers searching for specific keys in order to locate the related command files. In addition to search operations, the bots also frequently communicate with their peers and send *keep-alive* messages.

According to the botnet definition given before, a botnet is characterized by both a C&C communication channel (from which the botmaster's commands are received) and malicious activities (when commands are executed). Some other forms of malware (e.g., worms) may perform malicious activities, but they do not connect to a C&C channel. On the other hand, some normal applications (e.g., IRC clients and normal P2P file sharing software) may show communication patterns similar to a botnet's C&C channel, but they do not perform malicious activities.

In both centralized and P2P structures, bots within the same botnet are likely to behave similarly in terms of communication patterns. This is largely due to the fact that bots are non-human driven, pre-programmed to perform the same routine C&C logic/communication as coordinated by the same botmaster. In the centralized structure, even if the address of the C&C server may change frequently (e.g., by frequently changing the A record of a Dynamic DNS domain name), the C&C communication patterns remain unchanged. In the case of P2P-based botnets, the peer communications (e.g., to search for commands or to

send *keep-alive* messages) follow a similar pattern for all the bots in the botnet, although each bot may have a different set of neighbor peers and may communicate on different ports.

Regardless of the specific structure of the botnet (centralized or P2P), members of the same botnet (i.e., the bots) are coordinated through the C&C channel. In general, a bot*net* is different from a set of *isolated* individual malware instances, in which each different instance is used for a totally different purpose. Although in an extreme case a botnet can be configured to degenerate into a group of *isolated hosts*, this is not the common case. In this work, we focus on the most typical and useful situation in which bots in the same bot*net* perform similar/coordinated activities. To the best of our knowledge, this holds true for most of the existing botnets observed in the wild.

To summarize, we assume that bots within the same botnet will be characterized by similar malicious activities, as well as similar C&C communication patterns. Our assumption holds even in the case when the botmaster chooses to divide a botnet into *sub-botnets*, for example by assigning different tasks to different sets of bots. In this case, each sub-botnet will be characterized by similar malicious activities and C&C communications patterns, and our goal is to detect each sub-botnet. In Section 5.4 we provide a detailed discussion on possible *evasive* botnets that may violate our assumptions.

5.2 BotMiner: Architecture, Design and Implementation

5.2.1 BotMiner Architecture

Figure 20 shows the architecture of our BotMiner detection system, which consists of five main components: C-plane monitor, A-plane monitor, C-plane clustering module, A-plane clustering module, and cross-plane correlator.

The two traffic monitors in C-plane and A-plane can be deployed at the edge of the network examining traffic between internal and external networks, similar to BotHunter [46]

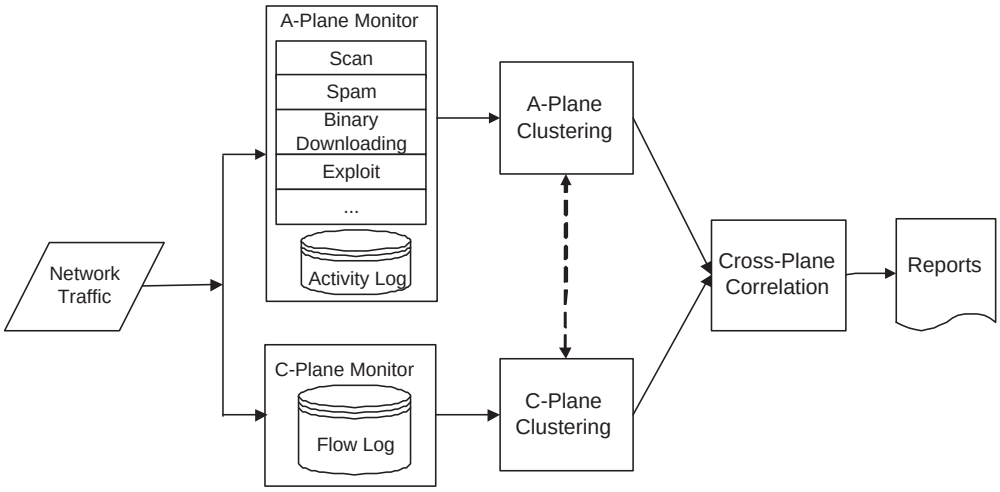


Figure 20: Architecture overview of our BotMiner detection framework.

and BotSniffer [48].[3] They run in parallel and monitor the network traffic. The C-plane monitor is responsible for logging network flows in a format suitable for efficient storage and further analysis, and the A-plane monitor is responsible for detecting suspicious activities (e.g., scanning, spamming, and exploit attempts). The C-plane clustering and A-plane clustering components process the logs generated by the C-plane and A-plane monitors, respectively. Both modules extract a number of features from the raw logs and apply clustering algorithms in order to find groups of machines that show very similar communication (in the C-plane) and activity (in the A-plane) patterns. Finally, the cross-plane correlator combines the results of the C-plane and A-plane clustering and makes a final decision on which machines are possibly members of a botnet. In an ideal situation, the traffic monitors should be distributed on the Internet, and the monitor logs are reported to a central repository for clustering and cross-plane analysis.

In our current prototype system, traffic monitors are implemented in C for the purpose of efficiency (working on real-time network traffic). The clustering and correlation analysis components are implemented mainly in Java and R (`http://www.r-project.org/`), and they work offline on logs generated from the monitors.

[3] All these tools can also be deployed in LANs.

The following sections present the details of the design and implementation of each component of the detection framework.

5.2.2 Traffic Monitors

C-plane Monitor. The C-plane monitor captures network flows and records information on *who is talking to whom*. Many network routers support the logging of network flows, e.g., Cisco (`www.cisco.com`) and Juniper (`www.juniper.net`) routers. Open source solutions like Argus (Audit Record Generation and Utilization System, `http://www.qosient.com/argus`) are also available. We adapted an efficient network flow capture tool developed at our research lab, i.e., `fcapture`,[4] which is based on the Judy library (`http://judy.sourceforge.net/`). Currently, we limit our interest to TCP and UDP flows. Each flow record contains the following information: time, duration, source IP, source port, destination IP, destination port, and the number of packets and bytes transfered in both directions. The main advantage of our tool is that it works very efficiently on high speed networks (very low packet loss ratio on a network with 300Mbps traffic), and can generate very compact flow records that comply with the requirement for further processing by the C-plain clustering module. As a comparison, our flow capturing tool generates compressed records ranging from 200MB to 1GB per day from the traffic in our academic network, whereas Argus generates around 36GB of compressed binary flow records per day on average (without recording any payload information). Our tool makes the storage of several weeks or even months of flow data feasible.

A-plane Monitor. The A-plane monitor logs information on *who is doing what*. It analyzes the outbound traffic through the monitored network and is capable of detecting several malicious activities that the internal hosts may perform. For example, the A-plane monitor is able to detect scanning activities (which may be used for malware propagation or DoS

[4]This tool will be released in open source soon.

attacks), spamming, binary downloading (possibly used for malware update), and exploit attempts (used for malware propagation or targeted attacks). These are the most common and "useful" activities a botmaster may command his bots to perform [24, 84, 126].

Our A-plane monitor is built based on Snort [86], an open-source intrusion detection tool, for the purpose of convenience. We adapted existing intrusion detection techniques and implemented them as Snort preprocessor plug-ins or signatures. For scan detection we adapted SCADE (Statistical sCan Anomaly Detection Engine), which is a part of BotHunter [46] and available at [27]. Specifically, we mainly use two anomaly detection modules: the *abnormally-high scan rate* and weighted *failed connection rate*. We use an `OR` combination rule, so that an event detected by either of the two modules will trigger an alert. In order to detect spam-related activities, we used a similar plug-in developed in BotSniffer [48]. We focused on detecting anomalous amounts of DNS queries for `MX` records from the same source IP and the amount of SMTP connections initiated by the same source to mail servers outside the monitored network. Normal clients are unlikely to act as SMTP servers and therefore should rely on the internal SMTP server for sending emails. Use of many distinct external SMTP servers for many times by the same internal host is an indication of possible malicious activities. For the detection of PE (Portable Executable) binary downloading we used an approach similar to PEHunter [118] and BotHunter's egg download detection method [46]. One can also use specific exploit rules in BotHunter to detect internal hosts that attempt to exploit external machines. Other state-of-the-art detection techniques can be easily added to our A-plane monitoring to expand its ability to detect typical botnet-related malicious activities.

It is important to note that A-plane monitoring alone is not sufficient for botnet detection purpose. First of all, these A-plane activities are not exclusively used in botnets. Second, because of our relatively loose design of A-plane monitor (for example, we will generate a log whenever there is a PE binary downloading in the network regardless of whether the binary is malicious or not), relying on only the logs from these activities will generate a lot

of false positives. This is why we need to further perform A-plane clustering analysis as discussed shortly in Section 5.2.4.

5.2.3 C-plane Clustering

C-plane clustering is responsible for reading the logs generated by the C-plane monitor and finding clusters of machines that share similar communication patterns. Figure 21 shows the architecture of the C-plane clustering.

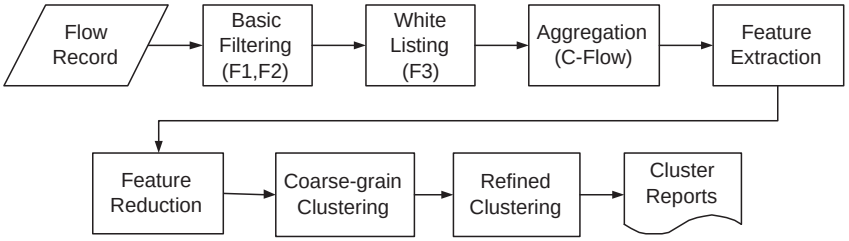


Figure 21: C-plane clustering.

First of all, we filter out irrelevant (or uninteresting) traffic flows. This is done in two steps: basic-filtering and white-listing. It is worth noting that these two steps are not critical for the proper functioning of the C-plane clustering module. Nonetheless, they are useful for reducing the traffic workload and making the actual clustering process more efficient. In the basic-filtering step, we filter out all the flows that are not directed from internal hosts to external hosts. Therefore, we ignore the flows related to communications between internal hosts[5] and flows initiated from external hosts towards internal hosts (filter rule 1, denoted as `F1`). We also filter out flows that are not completely established (filter rule 2, denoted as `F2`), i.e., those flows that only contain one-way traffic. These flows are mainly caused by scanning activity (e.g., when a host sends SYN packets without completing the TCP hand-shake). In white-list filtering, we filter out those flows whose destinations are well known as legitimate servers (e.g., `Google`, `Yahoo!`) that will unlikely host botnet C&C servers. This filter rule is denoted as `F3`. In our current evaluation, the white list is based

[5]If the C-plane monitor is deployed at the edge router, these traffic will not be seen. However, if the monitor is deployed/tested in a LAN, then this filtering can be used.

on the US top 100 and global top 100 most popular websites from `Alexa.com`.

After basic-filtering and white-listing, we further reduce the traffic workload by aggregating related flows into communication flows (C-flows) as follows. Given an epoch E (typically several hours), all m TCP/UDP flows that share the same protocol (TCP or UDP), source IP, destination IP and port, are aggregated into the same C-flow $c_i = \{f_j\}_{j=1..m}$, where each f_j is a single TCP/UDP flow. Basically, the set $\{c_i\}_{i=1..n}$ of all the n C-flows observed during E tells us "who was talking to whom," during that epoch.

5.2.3.1 Vector Representation of C-flows

The objective of C-plane clustering is to group hosts that share similar communication flows. This can be accomplished by clustering the C-flows. In order to apply clustering algorithms to C-flows we first need to translate them in a suitable vector representation. We extract a number of statistical features from each C-flow c_i, and translate them into d-dimensional pattern vectors $\vec{p_i} \in \mathbb{R}^d$. We can describe this task as a projection function F : C-plane $\rightarrow \mathbb{R}^d$. The projection function F is defined as follows. Given a C-flow c_i, we compute the discrete sample distribution of (currently) four random variables:

1. *the number of flows per hour (fph).* *fph* is computed by counting the number of TCP/IP flows in c_i that are present for each hour of the epoch E.

2. *the number of packets per flow (ppf).* *ppf* is computed by summing the total number of packets sent within each TCP/UDP flow in c_i.

3. *the average number of bytes per packets (bpp).* For each TCP/UDP flow $f_j \in c_i$ we divide the overall number of bytes transfered within f_j by the number of packets sent within f_j.

4. *the average number of bytes per second (bps).* *bps* is computed as the total number of bytes transfered within each $f_j \in c_i$ divided by the duration of f_j.

An example of the results of this process is shown in Figure 22, where we select a random client from a real network flow log (we consider a one-day epoch) and illustrate the features extracted from its visits to `Google`.

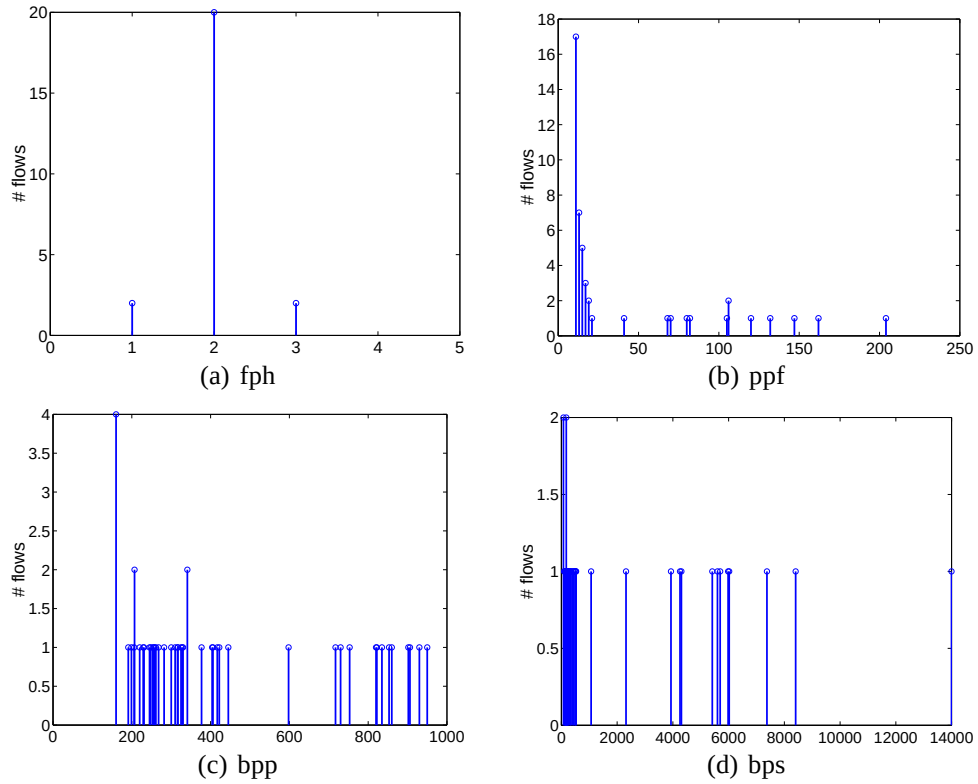


Figure 22: Visit pattern (shown in distribution) to `Google` from a randomly chosen normal client.

Given the discrete sample distribution of each of these four random variables, we compute an approximate version of it by means of a binning technique. For example, in order to approximate the distribution of *fph* we divide the x-axis in 13 intervals as $[0, k_1], (k_1, k_2], ..., (k_{12}, \infty)$. The values $k_1, .., k_{12}$ are computed as follows. First, we compute the overall discrete sample distribution of fph considering all the C-flows in the traffic for an epoch E. Then, we compute the quantiles[6] $q_{5\%}, q_{10\%}, q_{15\%}, q_{20\%}, q_{25\%}, q_{30\%}, q_{40\%}$, $q_{50\%}, q_{60\%}, q_{70\%}, q_{80\%}, q_{90\%}$, of the obtained distribution, and we set $k_1 = q_{5\%}$, $k_2 = q_{10\%}$, $k_3 = q_{15\%}$, etc. Now, for each C-flow we can describe its *fph* (approximate) distribution as

[6]The quantile $q_{l\%}$ of a random variable X is the value q for which $P(X < q) = l\%$.

a vector of 13 elements, where each element i represents the number of times *fph* assumed a value within the corresponding interval $(k_{i-1}, k_i]$. We also apply the same algorithm for *ppf*, *bpp*, and *bps*, and therefore we map each C-flow c_i into a pattern vector $\vec{p_i}$ of $d = 52$ elements. Figure 23 shows the scaled visiting pattern extracted form the same C-flow shown in Figure 22.

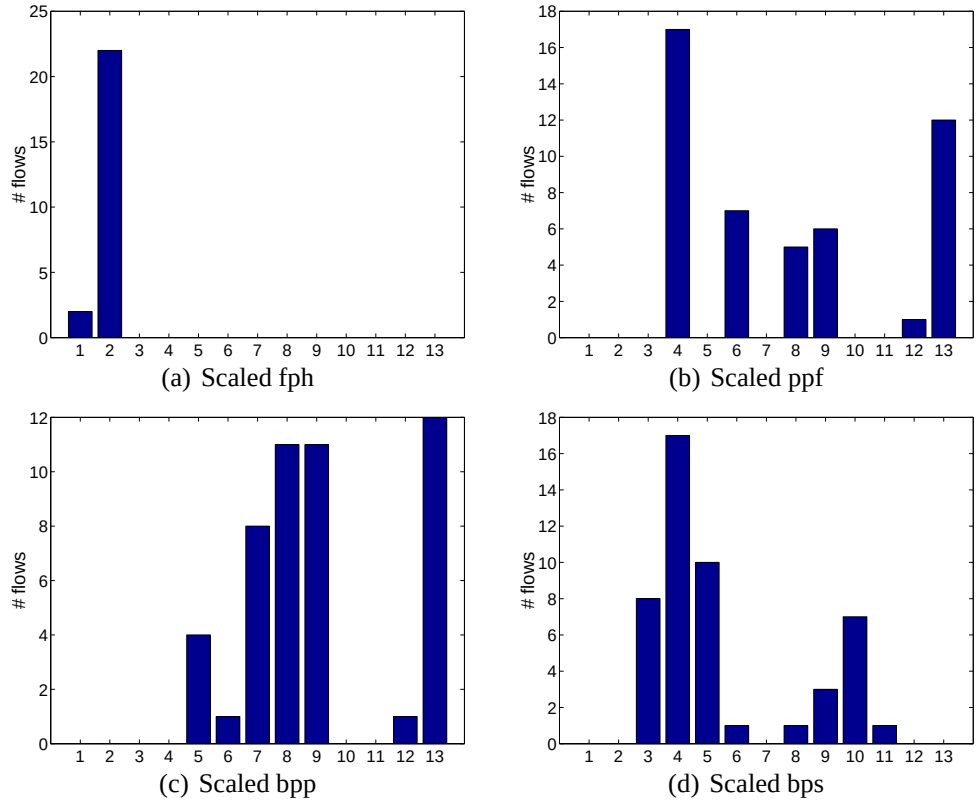


Figure 23: Scaled visit pattern (shown in distribution) to `Google` for the same client in Figure 22.

5.2.3.2 Two-step Clustering

Since bots belonging to the same botnet share similar behavior (from both the communication and activity points of view) as we discussed before, our objective is to look for groups of C-flows that are similar to each other. Intuitively, pattern vectors that are close to each other in $\mathbb{R}^d$ represent C-flows with similar communication patterns in the C-plane. For example, suppose two bots of the same botnet connect to two different C&C servers (because

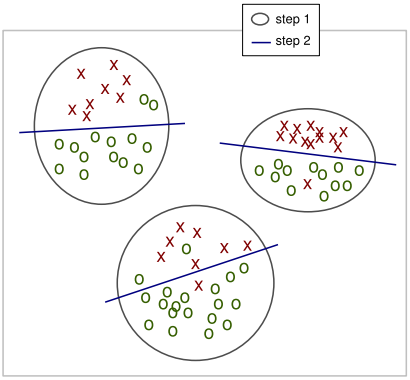


Figure 24: Two-step clustering of C-flows.

some botnets use multiple C&C servers). Although the connections from both bots to the C&C servers will be in different C-flows because of different source/destination pairs, their C&C traffic characteristics should be similar. That is, in $\mathbb{R}^d$, these C-flows should be found as being very similar. In order to find groups of hosts that share similar communication patterns, we apply clustering techniques on the dataset $\mathcal{D} = \{\vec{p_i} = F(c_i)\}_{i=1..n}$ of the pattern vector representations of C-flows. Clustering techniques perform unsupervised learning. Typically, they aim at finding meaningful groups of data points in a given feature space $\mathbb{F}$. The definition of "meaningful clusters" is application-dependent. Generally speaking, the goal is to group the data into clusters that are both compact and well separated from each other, according to a suitable similarity metric defined in the feature space $\mathbb{F}$ [55].

Clustering C-flows is a challenging task because $|\mathcal{D}|$, the cardinality of $\mathcal{D}$, is often large even for moderately large networks, and the dimensionality d of the feature space is also large. Furthermore, because the percentage of machines in a network that are infected by bots is generally small, we need to separate the few botnet-related C-flows from a large number of benign C-flows. All these make clustering of C-flows very expensive.

In order to cope with the complexity of clustering of $\mathcal{D}$, we solve the problem in several steps (currently in two steps), as shown in a simple form in Figure 24. At the first step, we perform coarse-grained clustering on a reduced feature space $\mathbb{R}^{d'}$, with $d' < d$, using a simple (i.e., non-expensive) clustering algorithm (we will explain below how we perform

dimensionality reduction). The results of this first-step clustering is a set $\{\mathcal{C}'_i\}_{i=1..\gamma_1}$ of γ_1 relatively large clusters. By doing so we subdivide the dataset $\mathcal{D}$ into smaller datasets (the clusters $\mathcal{C}'_i$) that contain "clouds" of points that are not too far from each other.

Afterwards, we refine this result by performing a second-step clustering on each different dataset $\mathcal{C}'_i$ using a simple clustering algorithm on the complete description of the C-flows in $\mathbb{R}^d$ (i.e., we do not perform dimensionality reduction in the second-step clustering). This second step generates a set of γ_2 smaller and more precise clusters $\{\mathcal{C}''_i\}_{i=1..\gamma_2}$.

We implemented first- and second-step clustering using the X-means clustering algorithm [75]. X-means is an efficient algorithm based on K-means [55], a very popular clustering algorithm. Different from K-means, the X-means algorithm does not require the user to choose the number K of final clusters in advance. X-means runs multiple rounds of K-means internally and performs efficient clustering validation using the Bayesian Information Criterion [75] in order to compute the best value of K. X-means is fast and scales well with respect to the size of the dataset [75].

For first-step (coarse-grained) clustering, we reduce the dimensionality of the feature space from $d = 52$ features (see Section 5.2.3.1) into $d' = 8$ features by simply computing the mean and the variance of the distribution of *fph*, *ppf*, *bpp*, and *bps* for each C-flow. Then we apply the X-means clustering algorithm on the obtained representation of C-flows to find the coarse-grained clusters $\{\mathcal{C}'_i\}_{i=1..\gamma_1}$. Since the size of the clusters $\{\mathcal{C}'_i\}_{i=1..\gamma_1}$ generated by first-step clustering is relatively small, we can now afford to perform a more expensive analysis on each $\mathcal{C}'_i$. Thus, for second-step clustering, we use all the $d = 52$ available features to represent the C-flows, and we apply the X-means clustering algorithm to refine the results of first-step clustering.

Of course, since unsupervised learning is a notoriously difficult task, the results of this two-step clustering algorithm may still be not perfect. As a consequence, the C-flows related to a botnet may be grouped into some distinct clusters, which basically represent *sub-botnets*. Furthermore, a cluster that contains mostly botnet or benign C-flows may

also contain some "noisy" benign or botnet C-flows, respectively. However, we would like to stress the fact that these problems are not necessarily critical and can be alleviated by performing correlation with the results of the activity-plane (A-plane) clustering (see Section 5.2.5).

Finally, we need to note that it is possible to bootstrap the clustering from A-plane logs. For example, one may apply clustering to only those hosts that appear in the A-plane logs (i.e., the suspicious activity logs). This may greatly reduce the workload of the C-plane clustering module, if speed is the main concern. Similarly, one may bootstrap the A-plane correlation from C-plane logs, e.g., by monitoring only clients that previously formed communication clusters, or by giving monitoring preference to those clients that demonstrate some persistent C-flow communications (assuming botnets are used for long-term purpose).

5.2.4 A-plane Clustering

In this stage, we perform two-layer clustering on activity logs. Figure 25 shows the clustering process in A-plane. For the whole list of clients that perform at least one malicious

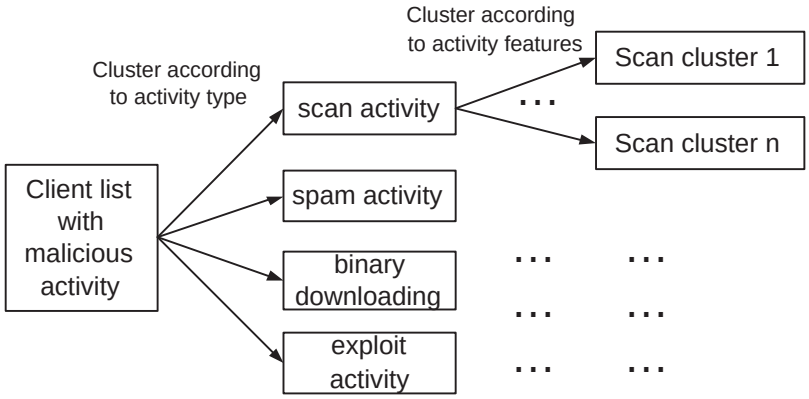


Figure 25: A-plane clustering.

activity during one day, we first cluster them according to the types of their activities (e.g., scan, spam, and binary downloading). This is the first layer clustering. Then, for each activity type, we further cluster clients according to specific activity features (the second

layer clustering). For the scan activity, features could include scanning ports, that is, two clients could be clustered together if they are scanning the same ports. Another candidate feature could be the target subnet/distribution, e.g., whether the clients are scanning the same subnet. For spam activity, two clients could be clustered together if their SMTP connection destinations are highly overlapped. This might not be robust when the bots are configured to use different SMTP servers in order to evade detection. One can further consider the spam content if the whole SMTP traffic is captured. To cluster spam content, one may consider the similarity of embedded URLs that are very likely to be similar with the same botnet [125], SMTP connection frequency, content entropy, and the normalized compression distance (NCD [14, 117]) on the entire email bodies. For outbound exploit activity, one can cluster two clients if they send the same type of exploit, indicated by the Snort alert SID. For binary downloading activity, two clients could be clustered together if they download similar binaries (because they download from the same URL as indicated in the command from the botmaster). A distance function between two binaries can be any string distance such as DICE used in [48].[7]

In our current implementation, we cluster scanning activities according to the destination scanning ports. For spam activity clustering, because there are very few hosts that show spamming activities in our monitored network, we simply cluster hosts together if they perform spamming (i.e., using only the first-layer clustering here). For binary downloading, we configure our binary downloading monitor to capture only the first portion (packet) of the binary for efficiency reasons (if necessary, we can also capture the entire binary). We simply compare whether these early portions of the binaries are the same or not. In other words, currently, our A-plane clustering implementation utilizes relatively weak cluster features. In the future, we plan to implement clustering on more complex

[7] In an extreme case that bots update their binaries from different URLs (and the binaries are packed to be polymorphic thereby different from each other), one should unpack the binary using tools such as Polyunpack [87] before calculating the distance. One may also directly apply normalized compression distance (NCD [14, 117]) on the original (maybe packed) binaries.

feature sets discussed above, which are more robust against evasion. However, even with the current weak cluster features, BotMiner already demonstrated high accuracy with a low false positive rate as shown in our later experiments.

5.2.5 Cross-plane Correlation

Once we obtain the clustering results from A-plane (activities patterns) and C-plane (communication patterns), we perform cross-plane correlation. The idea is to cross-check clusters in the two planes to find out intersections that reinforce evidence of a host being part of a botnet. In order to do this, we first compute a botnet score $s(h)$ for each host h on which we have witnessed at least one kind of suspicious activity. We filter out the hosts that have a score below a certain detection threshold θ, and then group the remaining most suspicious hosts according to a similarity metric that takes into account the A-plane and C-plane clusters these hosts have in common.

We now explain how the botnet score is computed for each host. Let H be the set of hosts reported in the output of the A-plane clustering module, and $h \in H$. Also, let $\mathcal{A}^{(h)} = \{A_i\}_{i=1..m_h}$ be the set of m_h A-clusters that contain h, and $\mathcal{C}^{(h)} = \{C_i\}_{i=1..n_h}$ be the set of n_h C-clusters that contain h. We compute the botnet score for h as

$$s(h) = \sum_{\substack{i,j \\ j>i \\ t(A_i)\neq t(A_j)}} w(A_i)w(A_j)\frac{|A_i \cap A_j|}{|A_i \cup A_j|} + \sum_{i,k} w(A_i)\frac{|A_i \cap C_k|}{|A_i \cup C_k|}, \quad (3)$$

where $A_i, A_j \in \mathcal{A}^{(h)}$ and $C_k \in \mathcal{C}^{(h)}$, $t(A_i)$ is the type of activity cluster A_i refers to (e.g., scanning or spamming), and $w(A_i) \geqslant 1$ is an *activity weight* assigned to A_i. $w(A_i)$ assigns higher values to "strong" activities (e.g., spam and exploit) and lower values to "weak" activities (e.g., scanning and binary download).

A host h will receive a high score if it has performed multiple types of suspicious activities, and if other hosts that are clustered with h also show the same multiple types of activities. For example, assume that h performed scanning and then attempted to exploit a machine outside the monitored network. Let A_1 be the cluster of hosts that were found to

perform scanning and were grouped with h in the same cluster. Also, let A_2 be a cluster related to exploit activities that includes h and other hosts that performed similar activities. A larger overlap between A_1 and A_2 would result in a higher score being assigned to h. Similarly, if h belongs to A-clusters that have a large overlap with C-clusters, then it means that the hosts clustered together with h share similar activities as well as similar communication patterns.

Given a predefined detection threshold θ, we consider all the hosts $h \in H$ with $s(h) > \theta$ as (likely) bots, and filter out the hosts whose scores do not exceed θ. Now, let $B \subseteq H$ be the set of detected bots, $\mathcal{A}^{(B)} = \{A_i\}_{i=1..m_B}$ be the set of A-clusters that each contains at least one bot $h \in B$, and $\mathcal{C}^{(B)} = \{C_i\}_{i=1..n_B}$ be the set of C-clusters that each contains at least one bot $h \in B$. Also, let $\mathcal{K}^{(B)} = \mathcal{A}^{(B)} \cup \mathcal{C}^{(B)} = \{K_i^{(B)}\}_{i=1..(m_B+n_B)}$ be an ordered union/set of A- and C-clusters. We then describe each bot $h \in B$ as a binary vector $b(h) \in \{0,1\}^{|\mathcal{K}^{(B)}|}$, where the i-th element $b_i = 1$ if $h \in K_i^{(B)}$, and $b_i = 0$ otherwise. Given this representation, we can define the following similarity between bots h_i and h_j as

$$sim(h_i, h_j) = \sum_{k=1}^{m_B} I(b_k^{(i)} = b_k^{(j)}) + I(\sum_{k=m_B+1}^{m_B+n_B} I(b_k^{(i)} = b_k^{(j)}) \geq 1), \tag{4}$$

where we use $b^{(i)} = b(h_i)$ and $b^{(j)} = b(h_j)$, for brevity. $I(X)$ is the indication function, which equals to one when the boolean argument X is true, and equals to zero when X is false. The intuition behind this metric is that if two hosts appear in the same activity clusters and in at least one common C-cluster, they should be clustered together.

This definition of similarity between hosts gives us the opportunity to apply hierarchical clustering. This allows us to build a dendrogram, i.e., a tree like graph (see Figure 26) that encodes the relationships among the bots. We use the Davies-Bouldin (DB) validation index [49] to find the best dendrogram cut, which produces the most compact and well separated clusters. The obtained clusters group bots in (sub-) botnets. Figure 26 shows a (hypothetical) example. Assuming that the best cut suggested by the DB index is the one at height 90, we would obtain two botnets, namely $\{h_8, h_3, h_5\}$, and $\{h_4, h_6, h_9, h_2, h_1, h_7\}$.

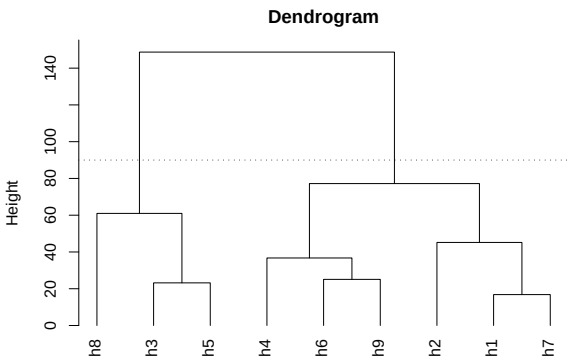


Figure 26: Example of hierarchical clustering for botnet detection.

In our current implementation, we simply set weight $w(A_i) = 1$ for all i and $\theta = 0$, which essentially means that we will consider all hosts that appear in two different types of A-clusters and/or in both A- and C-clusters as suspicious candidates for further hierarchical clustering.

5.3 Experiments

To evaluate our BotMiner detection framework and prototype system, we have tested its performance on several real-world network traffic traces, including both (presumably) normal data from our campus network and collected botnet data.

5.3.1 Experiment Setup and Data Collection

We set up traffic monitors to work on a span port mirroring a backbone router at the campus network of the College of Computing at Georgia Tech. The traffic rate is typically 100Mbps-300Mbps at daytime. We ran the C-plane and A-plane monitors for a continuous 10-day period in late 2007. A random sampling of the network trace shows that the traffic is very diverse, containing many normal application protocols, such as HTTP, SMTP, POP, FTP, SSH, NetBios, DNS, SNMP, IM (e.g., ICQ, AIM), P2P (e.g., Gnutella, Edonkey, BitTorrent), and IRC. This serves as a good background to test the false positives and detection performance on a normal network with rich application protocols.

We have collected a total of eight different botnets covering IRC, HTTP and P2P. Table 8 lists the basic information about these traces.

Table 8: Collected botnet traces in BotMiner evaluation, covering IRC, HTTP and P2P based botnets. Storm and Nugache share the same file, so the statistics of the whole file are reported.

Trace	Size	Duration	Pkt	Flows	Botnet clients	C&C server
Botnet-IRC-rbot	169MB	24h	1,175,083	180,988	4	1
Botnet-IRC-sdbot	66KB	9m	474	19	4	1
Botnet-IRC-spybot	15MB	32m	180,822	147,945	4	1
Botnet-IRC-N	6.4MB	7m	65,111	5635	259	1
Botnet-HTTP-1	6MB	3.6h	65,695	2,647	4	1
Botnet-HTTP-2	37MB	19h	395,990	9,716	4	1
Botnet-P2P-Storm	1.2G	24h	59,322,490	5,495,223	13	P2P
Botnet-P2P-Nugache	1.2G	24h	59,322,490	5,495,223	82	P2P

We re-used two IRC and two HTTP botnet traces introduced in [48], i.e., V-Spybot, V-Sdbot, B-HTTP-I, and B-HTTP-II. In short, V-Spybot and V-Sdbot are generated by executing modified bot code (Spybot and Sdbot [16]) in a fully controlled virtual network. They contain four Windows XP/2K IRC bot clients, and last several minutes. B-HTTP-I and B-HTTP-II are generated based on the description of Web-based C&C communications in [53, 96]. Four bot clients communicate with a controlled server and execute the received command (e.g., "`spam`"). In B-HTTP-I, the bot contacts the server periodically (about every five minutes) and the whole trace lasts for about 3.6 hours. In B-HTTP-II, we have a more stealthy C&C communication where the bot waits a random time between zero to ten minutes each time before it visits the server, and the whole trace lasts for 19 hours. These four traces are renamed as Botnet-IRC-spybot, Botnet-IRC-sdbot, Botnet-HTTP-1, and Botnet-HTTP-2, respectively. In addition, we also generated a new IRC botnet trace that lasts for a longer time (a whole day) using modified Rbot [12] source code. Again this is generated in a controlled virtual network with four Windows clients and one IRC server. This trace is labeled as Botnet-IRC-rbot.

We also obtained a real-world IRC-based botnet C&C trace that was captured in the wild in 2004, labeled as Botnet-IRC-N. The trace contains about 7-minute IRC C&C communications, and has hundreds of bots connected to the IRC C&C server. The botmaster set

the command "`.scan.startall`" in the TOPIC of the channel. Thus, every bot would begin to propagate through scanning once joining the channel. They report their successful transfer of binary to some machines, and also report the machines that have been exploited. We believe this could be a variant of Phatbot [16]. Although we obtained only the IRC C&C traffic, we hypothesize that the scanning activities are easy to detect given the fact that bots are performing scanning commands in order to propagate. Thus, we assume we have an A-plane cluster with the botnet members because we want to see if we can still capture C-plane clusters and obtain cross-plane correlation results.

Finally, we obtained a real-world trace containing two P2P botnets, Nugache [64] and Storm [44, 52]. The trace lasts for a whole day, and there are 82 Nugache bots and 13 Storm bots in the trace. It was captured from a group of honeypots running in the wild in late 2007. Each instance is running in Wine (an open source implementation of the Windows API on top of Unix/Linux) instead of a virtual or physical machine. Such a set-up is known as winobot [29] and is used by researchers to track botnets. By using a lightweight emulation environment (Wine), winobots can run hundreds and thousands of black-box instances of a given malware. This allows one to participate in a P2P botnet *en mass*. Nugache is a TCP-based P2P bot that performs encrypted communications on port 8. Storm, originating in January of 2007, is one of the very few known UDP-based P2P bots. It is based on the Kademlia [68] protocol and makes use of the Overnet network [8] to locate related data (e.g., commands). Storm is well-known as a spam botnet with a huge number of infected hosts [61]. In the implementation of winobot, several malicious capabilities such as sending spam are disabled for legality reason, thus we can not observe spam traffic from the trace. However, we ran a full version of Storm on a VM-based honeypot (instead of Wine environment) and easily observed that it kept sending a huge amount of spam traffic, which makes the A-plane monitoring quite easy. Similarly, when running Nugache on a VM-based honeypot, we observed scanning activity to port 8 because it attempted to connect to its seeding peers but failed a lot of times (because the peers may not be

available). Thus, we can detect and cluster A-plane activities for these P2P botnets.

5.3.2 Evaluation Results

Table 9 lists the statistics for the 10 days of network data we used to validate our detection system. For each day there are around 5-10 billion packets (TCP and UDP) and 30-100 million flows. Table 9 shows the results of several steps of filtering. The first step of filtering (filter rule `F1`) seems to be the most effective filter in terms of data volume reduction. `F1` filters out those flows that are not initiated from internal hosts to external hosts, and achieves about 90% data volume reduction. The is because most of the flows are within the campus network (i.e., they are initiated from internal hosts towards other internal hosts). `F2` further filters out around 0.5-3 million of non-completely-established flows. `F3` further reduces the data volume by filtering out another 30,000 flows. After applying all the three steps of filtering, there are around 1 to 3 million flows left per day. We converted these remaining flows into C-flows as described in Section 5.2.3, and obtained around 40,000 TCP C-flows and 130,000 UDP C-flows per day.

Table 9: C-plane traffic statistics, basic results of filtering, and C-flows in BotMiner evaluation.

Trace	Pkts	Flows	Filtered by `F1`	Filtered by `F2`	Filtered by `F3`	Flows after filtering	C-flows (TCP/UDP)
Day-1	5,178,375,514	23,407,743	20,727,588	939,723	40,257	1,700,175	66,981 / 132,333
Day-2	7,131,674,165	29,632,407	27,861,853	533,666	25,758	1,211,130	34,691 / 96,261
Day-3	9,701,255,613	30,192,645	28,491,442	513,164	24,329	1,163,710	39,744 / 94,081
Day-4	14,713,667,172	35,590,583	33,434,985	600,901	33,958	1,520,739	73,021 / 167,146
Day-5	11,177,174,133	56,235,380	52,795,168	1,323,475	40,016	2,076,721	57,664 / 167,175
Day-6	9,950,803,423	75,037,684	71,397,138	1,464,571	51,931	2,124,044	59,383 / 176,210
Day-7	10,039,871,506	109,549,192	105,530,316	1,614,158	56,688	2,348,030	55,023 / 150,211
Day-8	11,174,937,812	96,364,123	92,413,010	1,578,215	60,768	2,312,130	56,246 / 179,838
Day-9	9,504,436,063	62,550,060	56,516,281	3,163,645	30,581	2,839,553	25,557 / 164,986
Day-10	11,071,701,564	83,433,368	77,601,188	2,964,948	27,837	2,839,395	25,436 / 154,294

We then performed two-step clustering on C-flows as described in Section 5.2.3. Table 10 shows the clustering results and false positives (number of clusters that are not botnets). The results for the first 5 days are related to both TCP and UDP traffic, whereas in the last 5 days we focused on only TCP traffic.

It is easy to see from Table 10 that there are thousands of C-clusters generated each day. In addition, there are several thousand activity logs generated from A-plane monitors. Since we use relatively weak monitor modules, it is not surprising that we have this many activity

Table 10: C-plane and A-plane clustering results in BotMiner evaluation.

Trace	Step-1 C-clusters	Step-2 C-clusters	A-plane logs	A-clusters	False Positive Clusters	FP Rate
TCP/UDP						
Day-1	1,374	4,958	1,671	1	0	0 (0/878)
Day-2	904	2,897	5,434	1	1	0.003 (2/638)
Day-3	1,128	2,480	4,324	1	1	0.003 (2/692)
Day-4	1,528	4,089	5,483	4	4	0.01 (9/871)
Day-5	1,051	3,377	6,461	5	2	0.0048 (4/838)
TCP only						
Day-6	1,163	3,469	6,960	3	2	0.008 (7/877)
Day-7	954	3,257	6,452	5	2	0.006 (5/835)
Day-8	1,170	3,226	8,270	4	2	0.0091 (8/877)
Day-9	742	1,763	7,687	2	0	0 (0/714)
Day-10	712	1,673	7,524	0	0	0 (0/689)

logs. Many logs report binary downloading events or scanning activities. We cluster these activity logs according to their activity features. As explained earlier, we are interested in groups of machines that perform activities in a similar/coordinated way. Therefore, we filter out the A-clusters that contain only one host. This simple filtering rule allows us to obtain a small number of A-clusters and reduce the overall false positive rate of our botnet detection system.

Afterwards, we apply cross-plane correlation. We assume that the traffic we collected from our campus network is normal. In order to verify this assumption we used state-of-the-art botnet detection techniques like BotHunter [46] and BotSniffer [48]. Therefore, any cluster generated as a result of the cross-plane correlation is considered as a *false positive cluster*. It is easy to see from Table 10 that there are very few such false positive clusters every day (from zero to four). Most of these clusters contain only two clients (i.e., they induce two false positives). In three out of ten days no false positive was reported. In both Day-2 and Day-3, the cross-correlation produced one false positive cluster containing two hosts. Two false positive clusters were reported in each day from Day-5 to Day-8. In Day-4, the cross-plane correlation produced four false positive clusters.

For each day of traffic, the last column of Table 10 shows the false positive rate (FP rate), which is calculated as the fraction of IP addresses reported in the false positive clusters over the total number of distinct normal clients appearing in that day. After further

analysis we found that many of these false positives are caused by clients performing binary downloading from websites not present in our whitelist. In practice, the number of false positives may be reduced by implementing a better binary downloading monitor and clustering module, e.g., by capturing the entire binary and performing content inspection (using either anomaly-based detection systems [88] or signature-based AV tools).

In order to validate the detection accuracy of BotMiner, we overlaid botnet traffic to normal traffic. We consider one botnet trace at a time and overlay it to the entire normal traffic trace of Day-2. We simulate a near-realistic scenario by constructing the test dataset as follows. Let n be the number of distinct bots in the botnet trace we want to overlay to normal traffic. We randomly select n distinct IP addresses from the normal traffic trace and map them to the n IP addresses of the bots. That is, we replace an IP_i of a normal machine with the IP_i of a bot. In this way, we obtain a dataset of mixed normal and botnet traffic where a set of n machines show both normal and botnet-related behavior. Table 11 reports the detection results for each botnet.

Table 11: Botnet detection results using BotMiner.

Botnet	Number of Bots	Detected?	Clustered Bots	Detection Rate	False Positive Clusters/Hosts	FP Rate
IRC-rbot	4	YES	4	100%	1/2	0.003
IRC-sdbot	4	YES	4	100%	1/2	0.003
IRC-spybot	4	YES	3	75%	1/2	0.003
IRC-N	259	YES	258	99.6%	0	0
HTTP-1	4	YES	4	100%	1/2	0.003
HTTP-2	4	YES	4	100%	1/2	0.003
P2P-Storm	13	YES	13	100%	0	0
P2P-Nugache	82	YES	82	100%	0	0

Table 11 shows that BotMiner is able to detect all eight botnets. We verified whether the members in the reported clusters are actually bots or not. For 6 out of 8 botnets, we obtained 100% detection rate, i.e., we successfully identified all the bots within the 6 botnets. For example, in the case of P2P botnets (Botnet-P2P-Nugache and Botnet-P2P-Storm), BotMiner correctly generated a cluster containing all the botnet members. In the case of Botnet-IRC-spybot, BotMiner correctly detected a cluster of bots. However, one of the bots belonging to the botnet was not reported in the cluster, which means that the detector generated a false negative. Botnet-IRC-N contains 259 bot clients. BotMiner was able to

identify 258 of the bots in one cluster, whereas one of the bots was not detected. Therefore, in this case BotMiner had a detection rate of 99.6%.

There were some cases in which BotMiner also generated a false positive cluster containing two normal hosts. We verified that these two normal hosts in particular were also responsible for the false positives generated during the analysis of the Day-2 normal traffic (see Table 10).

As we can see, BotMiner performs quite well in our experiments, showing a very high detection rate with relatively few false positives in real-world network traces.

5.4 Limitations and Potential Solutions

Like any intrusion/anomaly detection system, BotMiner is not perfect or complete. It can have false positives similar to the cases of BotSniffer because they both use horizontal correlation. It is also likely that once adversaries know our detection framework and implementation, they might find some ways to evade detection, e.g., by evading the C-plane and A-plane monitoring and clustering, or the cross-plane correlation analysis. We now address these limitations and discuss possible solutions.

5.4.1 Evading C-plane Monitoring and Clustering

Botnets may try to utilize a legitimate website (e.g., `Google`) for their C&C purpose in attempt to evade detection. Evasion would be successful in this case if we whitelisted such legitimate websites to reduce the volume of monitored traffic and improve the efficiency of our detection system. However, if a legitimate website, say `Google`, is used as a means to locate a secondary URL for actual command hosting or binary downloading, botnets may not be able to hide this secondary URL and the corresponding communications. Therefore, clustering of network traffic towards the server pointed by this secondary URL will likely allow us to detect the bots. Also, whitelisting is just an optional operation. One may easily choose not to use whitelisting to avoid such kind of evasion attempts (of course, in this case one may face the tradeoff between accuracy and efficiency).

Botnet members may attempt to intentionally manipulate their communication patterns to evade our C-plane clustering. The easiest thing is to switch to multiple C&C servers. However, this does not help much to evade our detection because such peer communications could still be clustered together just like how we cluster P2P communications. A more advanced way is to randomize each individual communication pattern, for example by randomizing the number of packets per flow (e.g., by injecting random packets in a flow), and the number of bytes per packet (e.g., by padding random bytes in a packet). However, such randomization may introduce similarities among botnet members if we measure the distribution and entropy of communication features. Also, this randomization may raise suspicion because normal user communications may not have such randomized patterns. Advanced evasion may be attempted by bots that try to mimic the communication patterns of normal hosts, in a way similar to polymorphic blending attacks [38]. Furthermore, bots could use covert channes [2] to hide their actual C&C communications. We acknowledge that, generally speaking, communication randomization, mimicry attacks and covert channel represent limitations for all traffic-based detection approaches, including BotMiner's C-plane clustering technique. By incorporating more detection features such as content inspection and host level analysis, the detection system may make evasion more difficult.

Finally, we note that if botnets are used to perform multiple tasks (in A-plane), we may still detect them even when they can evade C-plane monitoring and analysis. By using the scoring algorithm described in Section 5.2.5, we can perform cross clustering analysis among multiple activity clusters (in A-plane) to accumulate the suspicious score needed to claim the existence of botnets. Thus, we may even *not* require C-plane analysis if there is already a *strong* cross-cluster correlation among different types of malicious activities in A-plane. For example, if the same set of hosts involve several types of A-plane clusters (e.g., they send spams, scan others, and/or download the same binaries), they can be reported as botnets because those events together are highly suspicious and most likely indicating botnet behavior [46, 48].

5.4.2 Evading A-plane Monitoring and Clustering

Malicious activities of botnets are unlikely or relatively hard to change as long as the botmaster wants the botnets to perform "useful" tasks. However, the botmaster can attempt to evade BotMiner's A-plane monitoring and clustering in several ways.

Botnets may perform very stealthy malicious activities in order to evade the detection of A-plane monitors. For example, they can scan very slowly (e.g., send one scan per hour), send spam very slowly (e.g., send one spam per day). This will evade our monitor sensors. However, this also puts a limit on the utility of bots.

In addition, as discussed above, if the botmaster commands each bot *randomly and individually* to perform different task, the bots are not different from previous generations of isolated, individual malware instances. This is unlikely the way a bot*net* is used in practice. A more advanced evasion is to differentiate the bots and avoid commanding bots in the same monitored network the same way. This will cause additional effort and inconvenience for the botmaster. To defeat such an evasion, we can deploy distributed monitors on the Internet to cover a larger monitored space.

Note, if the botmaster takes the extreme action of randomizing/individualizing both the C&C communications and attack activities of each bot, then these bots are probably not part of a bot*net* according to our specific definition because the bots are not performing similar/coordinated commanded activities. Orthogonal to the *horizontal correlation* approaches such as BotMiner to detect a bot*net*, we can always use complementary systems like BotHunter [46] that examine the behavior history of *distinct* host for a dialog (vertical) correlation-based approach to detect *individual* bots.

5.4.3 Evading Cross-plane Analysis

A botmaster can command the bots to perform an extremely delayed task (e.g., delayed for days after receiving commands). Thus, the malicious activities and C&C communications are in different days. If only using one day's data, we may not be able to yield cross-plane

clusters. As a solution, we may use multiple-day data and cross check back several days. Although this has the hope of capturing these botnets, it may also suffer from generating more false positives. Clearly, there is a tradeoff. The botmaster also faces the tradeoff because a very slow C&C essentially impedes the efficiency in controlling/coordinating the bot army. Also, a bot-infected machine may be disconnected from the Internet or be powered off by the users during the delay and become unavailable to the botmaster.

In short, while it is possible that a botmaster can find a way to exploit the limitations of BotMiner, the convenience or the efficiency of botnet C&C and the utility of the botnet also suffer. Thus, we believe that our protocol- and structure-independent detection framework represents a significant advance in botnet detection. In our future work, we will study new techniques to monitor/cluster communication and activity patterns of botnets, and these techniques are intended to be more robust to evasion attempts.

5.5 Summary

Botnet detection is a challenging problem. In this chapter, we proposed a novel network anomaly-based botnet detection system that is independent of the protocol and structure used by botnets. Our system exploits the essential definition and properties of botnets, i.e., bots within the same botnet will exhibit similar C&C communication patterns and similar malicious activity patterns. In our experimental evaluation on many real-world network traces, BotMiner shows excellent detection accuracy on various types of botnets (including IRC-based, HTTP-based, and P2P-based botnets) with a very low false positive rate on normal traffic. BotMiner is promising because it can potentially resist the evolution and changes in botnet C&C techniques in the future.

CHAPTER VI

BOTPROBE: CAUSE-EFFECT CORRELATION-BASED BOTNET C&C DETECTION USING ACTIVE TECHNIQUES

We have introduced BotHunter, BotSniffer, and BotMiner. These three systems use a passive strategy to monitor the network traffic. A limitation of such a passive approach is the relatively longer time in collecting enough evidence for detection. For example, BotMiner is an offline correlation technique that performs analysis usually on one day's C-plane data. BotSniffer is quicker than BotMiner, but generally may require observing several rounds/instances of message/activity responses for enough confidence of spatial-temporal correlation.[1] BotHunter tracks the bot infection dialog and requires observing multiple stages. The main reason for the aforementioned systems to take a relatively long time for detection is because they *passively* wait to observe enough events/evidences. It is possible that these events may occur infrequently. For example, for an IRC-based botnet, the actual C&C interaction is rare because the botmaster cannot always be online to command the bot army.

In this chapter, we study botnet detection using active techniques to *actively* collect evidences and thereby shorten the monitor and detection time, and present a new botnet detection system, BotProbe. We want to answer the following questions: Assume we observe only one round of botnet C&C interaction,[2] can we still detect bots with a high probability? What if we observe *zero* round of interaction? We will show that BotProbe can achieve the detection goal for many real-world botnets that use chatting-like C&C protocols such as IRC, and improve the effectiveness and efficiency compared to existing techniques in

[1] If there are enough bots monitored, BotSniffer only needs to observe one round.

[2] One round of C&C interaction is defined as a typical command-then-response interaction. We further clarify this command-response pattern of botnet C&C and various types of response in Section 6.1.

many cases.

Key Observation: A typical way to detect bots by observing one C&C communication interaction is to use a signature-based approach. However, such an approach is complicated by a recent trend among bots to use an obfuscated (obscure) C&C communication. In this chapter, we explore the feasibility of using *active* techniques to assist with the C&C detection challenge. We posit that instead of passively inspecting two-way network flows, one could engage in the active manipulation of selected suspicious sessions to better identify botnet dialog. Our detection strategy, which we call *botnet probing*, is based on two observations. First, a typical botnet C&C interaction has a clear command-response pattern, and thereby a stateless bot will tend to behave deterministically[3] to dialog replays, whereas interaction with a human-controlled end point will be nondeterministic. Second, bots are *pre-programmed* to respond to the set of commands they receive, and unlike humans, bots have limited tolerance for typographical errors in conversations (aka the Turing test [101]).

New Approach and System: In this work, we focus on a specific class of botnets using chatting-like C&C protocols such as IRC, which is the most widely used C&C protocol in present botnets. Based on the above observations, we develop a set of active probing techniques to detect stateless botnet communications, regardless of whether the botnet communications are protected using obfuscation. At a first glance, these active techniques may be aggressive and controversial because of the interference they may introduce to normal benign communications/chatting. While a legitimate concern, we propose to ameliorate this interference in multiple ways. First, we provide a set of candidate filters that use heuristics to filter out a large class of well-behaved connections. Second, we provide a hypothesis testing framework that enables network administrators to tune the level of expected interference with detection rates. Finally, we argue that limited interference might be acceptable in pure IRC-like chatting channels on which no critical applications are built, and certain deployments such as military scenarios, particularly if users are educated about

[3]Examination of popular bot source code and binaries reveals that most bot communications are stateless.

the presence of such probing monitors. We develop the BotProbe prototype system to demonstrate this active technique. By actively probing botnets for several times, we can accumulate enough evidence of cause-effect correlation caused by the command-response pattern of botnet C&C. We only need to observe one or even zero round of actual C&C interaction before probing. Thus, we can greatly shorten the detection time compared to a passive approach.

Contributions:

- We propose active botnet probing based on cause-effect correlation as a novel approach to complement existing botnet C&C detection.
- We present a hypothesis testing framework for detecting deterministic communication patterns. This helps us to achieve bounded false positive and false negative rates.
- We develop BotProbe, a prototype system implementation of the framework that validates our approach with contemporary IRC-based bots such as Sdbot, Phatbot, Rbot, RxBot, Agobot, Wargbot, and IRCBot.
- We show with a real-world example that BotProbe can also assist with automating a chosen-ciphertext attack to break the encryption of some botnet C&C.
- We conduct a real user study on approximately 100 users to evaluate false positive rates.

Chapter organization: In Sections 6.1, we present the problem statement and our assumptions. In Section 6.2, we provide an overview of the architecture, and describe our probing techniques and detection algorithms. In Section 6.3, we present our prototype system and experimental results with botnet probing. We discuss concerns and limitations in Section 6.4 and summarize our findings in Section 6.5.

6.1 Problem Statement and Assumptions

A unique property of a botnet that separates it from other malware families is the command and control (C&C) channel, which the botmaster uses to command the bot army to perform different tasks. The detection of this C&C channel is fundamental to the identification of compromised victims and tear down of C&C servers. Although bot developers have the option of devising novel protocols for C&C, most contemporary bot C&C communications are overlaid onto existing client-server protocols such as IRC. This prevailing tendency to overlay botnet C&Cs on *existing* protocols may have several plausible explanations: (a) existing intrusion detection systems are trivial to evade (e.g., using obfuscation schemes discussed shortly) and have not provided sufficient incentive for botnets to innovate; (b) existing protocols provide greater flexibility in using available server software and installations; (c) existing protocols invoke less suspicion than neoteric protocols.

In this work, we limit our focus on botnet C&Cs using *chatting-like* protocols such as IRC and Instant Messaging [72], which count for a large portion of contemporary botnets. IRC is still the most prevailing communication channel among botnets. Except for a few HTTP botnets (e.g., Bobax [96]) and P2P botnets (e.g., Nugache [64] and Storm [44]), most of the discovered bot*nets* are IRC-based. In [72], the feasibility of using some instant messaging protocols such as AIM is also discussed.

While botnets still chooses to use chatting-like communication (e.g., IRC) as the underlying C&C protocol, the message/content portion of the conversations can be obfuscated (e.g., using a custom dialect, a foreign language, or a naive encryption technique such as simple XOR, substitution or hashing) to evade signature-based intrusion detection systems and to protect botnet C&Cs from being spied by honeypot-based tracking approaches [82]. Actually, we have observed a lot of new generation of IRC botnets utilizing such obscure C&Cs (real-world examples will be showed in Section 6.3). Existing botnet detection approaches rely heavily on signatures, which are of limited value when even unobfuscated C&C protocols and key word exchanges are easily and rapidly altered. The need for *new*

methods to detect both obfuscated and rapidly changing C&C communication channels is dire.

Behavior-based detection approaches such as our BotHunter, BotSniffer, and BotMiner can detection botnets through behavior anomaly. Nevertheless, as discussed earlier, systems like BotHunter still use signature-based component for C&C communication detection. More importantly, these passive monitoring systems usually require a relatively longer time to observe sufficient communication/activity/behavior for accurate detection. Real-world IRC-based botnet C&C communications, however, usually can be quite, i.e., they have infrequent C&C interactions because the botmaster is not always online. Our goal here is to evaluate the feasibility of detecting botnet C&C channels using *active* network traffic inspection strategies, given observing only limited number of C&C interactions, thereby to shorten the detection time. By active, we mean that we assess traffic for suspicious traffic sessions, which may lead us to dynamically inject packets that will probe the internal client to determine whether that side of the communicating/chatting session is being managed by a human or a bot.

To achieve the goal, first we need to examine the *invariant* that can be used to differentiate a bot from human chatting. We observe that bots are *pre-programmed to respond* to certain received commands, and these responses are consistent across command repetition. Different from normal human chatting, the above command-response pattern has a strong cause-effect correlation, i.e., the command causes the response in a deterministic way. This is the key intuition we use in designing our algorithm. In addition, we observe that bots are different from humans in tolerating typographical errors, i.e., if the command is altered by even one character, bots are not likely to process the command properly. This auxiliary intuition helps us design one of our detection algorithms. Before introducing our algorithms and system in detail, we present the adversary model, i.e., or more precisely, the detailed communication patterns that we seek to identify when adversaries communicate with compromised machines inside our network perimeter.

Adversary Assumption: Botnet C&C communications are well-structured duplex flows, similar to a command-response protocol, i.e., a bot should respond when it receives a predefined command in a reasonable time. The network-level response of a bot to an (obfuscated) command might be either message response or activity response, or both [48]. A typical example of message response is an IRC `PRIVMSG` message. For example, when the botmaster issues a "`.sysinfo`" command,[4] each bot replies with a `PRIVMSG` message telling its host system information, such as CPU, memory, and software version. There are three most common activity responses: scan response (bot performs network scan or DoS attack), third-party access (e.g., bot connects to a certain address to download/update its binary), and spam response (bot sends spams). For instance, when the botmaster issues a scanning command (e.g., "`.scan.startall`"), the bots usually perform network scanning and reply with the scanning progress and/or any new victims they have infected. This involves both an activity response (scan) and a message response. One may define other possible responses, but from our observation of live bot infections, these aforementioned types of responses are highly representative and regularly encountered.

Fortunately, the assumption of command-response pattern holds in almost all existing botnets, because the botmaster needs the bots to perform some (malicious) activity, and usually requires feedback to track the bot-infected machine information and execution progress/result from its bot army. Thus, we can observe message/activity responses corresponding to most botnet commands. According to a honeynet technical report [126], about 53% of botnet commands observed in thousands of real-world IRC-based botnets are scan-related (for propagation or reconnaissance) and about 14.4% are related to binary download (for malware update). Also, many HTTP-based botnets are primarily used for sending spam [96]. Thus, for most infections, we can expect to observe (malicious) activity responses with a high probability [24].

[4] We assume the botmaster could obfuscate the C&C channel using simple encryption or substitution, e.g., say "`hello`" instead of "`.sysinfo`."

Detection Assumption: We now discuss the design assumptions used in defining our architecture for actively probing and detecting botnet C&C channels:

- *Input Perspective.* Our assumed solution will reside at the network egress point (as a middlebox), where it can observe all flows that cross the network perimeter. Furthermore, the system is in-line with the communication, and has the authority to inject or modify *inbound* packets, as necessary.

- *Chatting Protocol Awareness.* Our solution incorporates knowledge of the standard (chatting) protocols that botnets use to overlay their C&C communications. For example, in the case of IRC-based bots, we can comprehend IRC keywords and `PRIVMSG` message exchanges.

- *C&C Grammar Agnostic.* We do not assume an *a priori* understanding of the syntax and semantics of the botnet C&C grammar that is overlaid on standard network protocols.

6.2 Active Botnet Probing: Architecture and Algorithms

6.2.1 Architecture Design

Our botnet C&C detection architecture has two integral components, as shown in Figure 27.

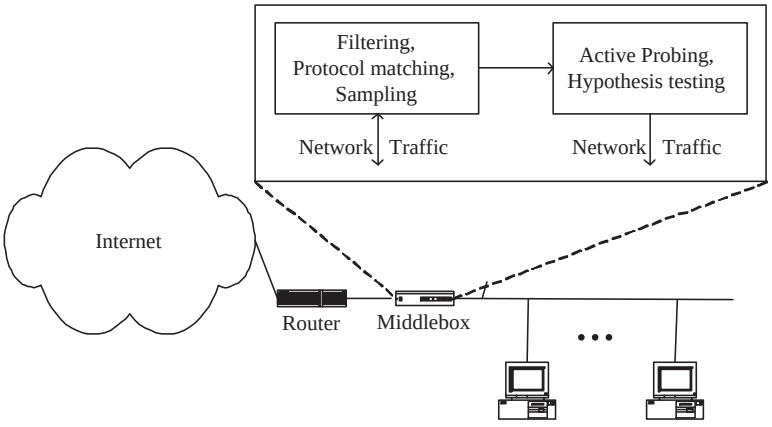


Figure 27: Two-layer architecture of using active techniques for identifying botnet C&Cs.

The first component performs benign traffic filtering, protocol-matching (selects protocols often exploited for C&C transmissions, e.g., IRC), and flow sampling. Thus, it leaves only a small portion of highly suspicious candidates worthy of deeper examination. Benign (chatting-like) traffic filtering modules can be implemented using a general traffic feature vector (e.g., duration of the flow, average bytes per packet, average bytes per second) similar to [59, 66, 98]. Or, in the case of IRC-based C&C detection, we can use the following protocol matching policies to perform detection in a port-independent fashion:

1. A traffic filter removes non-TCP flows.

2. Port-independent IRC protocols are keyword matched, e.g., "`NICK`," "`USER`," "`PRIVMSG`." This analysis occurs on the first few packets of established TCP flows (which indicate the beginning of an IRC session [5]).

3. A volume filter that mainly focuses on infrequent chatting IRC channels (because overly chatty IRC channels are unlikely to be used for botnet C&C).

4. A message filter finds a candidate list of command-like packets (IRC `PRIVMSG` and IRC `TOPIC`) that can cause client responses.

Once we have completed the above down-selection to our candidate flows, we then focus our analyses on the `TOPIC` and `PRIVMSG` message packets, where the overlay C&C commands/responses typically reside. In addition, one can incorporate any other behavior-based logic into these filters.

The second component implements what we refer to as our *BotProbe analysis* scheme. To illustrate the scheme, let us suppose we have a candidate suspicious IRC session and we need to further identify whether there is another layer of overlay C&C-like protocol. We observe a command-then-response-like packet pair (P_c, P_r) where P_c is a short packet from the server, and P_r is a response from the client immediately after the receiving of

P_c.[5] We hypothesize that this command-response pattern is from a bot instead of a human. However, observing only this likely command-response pair is not enough to make the claim, because it could be caused by chance.[6] We want to make sure whether there is truly a cause-effect correlation between the command and the response, which is a distinguishing feature between botnet C&C and human chatting. To achieve the detection goal with high accuracy, we perform several rounds of active probing and use a sequential hypothesis testing technique to obtain enough confidence. Next section will detail the design space of active probing techniques.

6.2.2 Design Choices of Active Probing Techniques

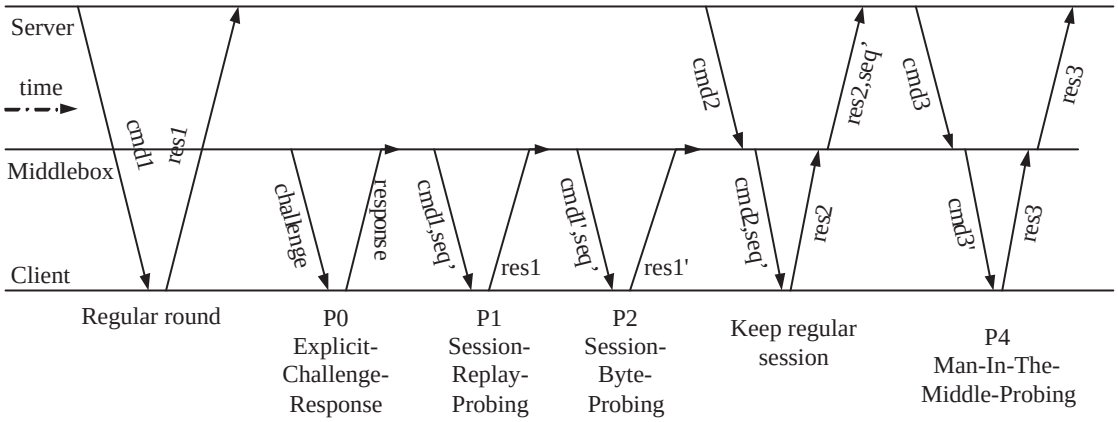
We investigate the design choices of active probing strategies and illustrate several probing techniques in Figure 28. This is by no means a complete list, but provides a good coverage and demonstration of active probing techniques. BotProbe can use one or a combination of these techniques.

P0 (Explicit-Challenge-Response). An example explicit-validation mechanism is one in which educated users participate in the BotProbe scheme knowingly. For example, a BotProbe system may prompt users to perform a reverse Turing test, when a new IRC session among two IP addresses is first encountered by BotProbe. The in-line monitor could request that the internal human IRC participant visit a website to read and translate a CAPTCHA [107]. Alternatively, BotProbe can inject a simple puzzle for the internal participant to solve. Although simple and effective, such a technique requires user awareness, compliance, and tolerance to be successful. We further discuss our experience of this technique in an actual user study in Section 6.3.3.

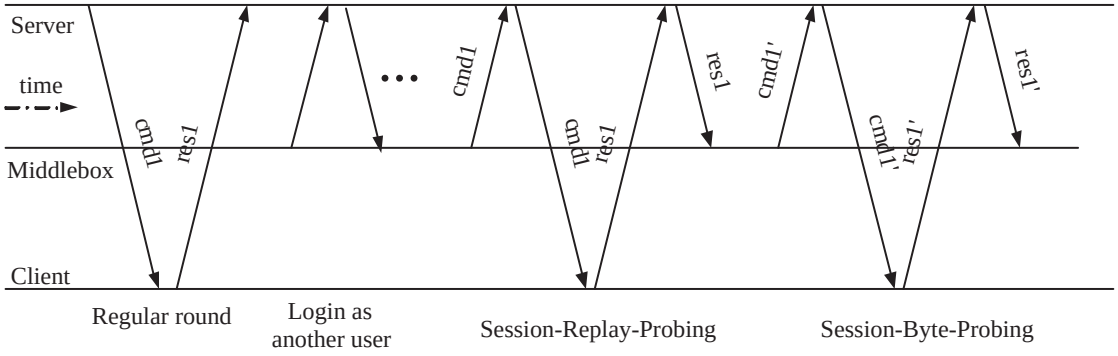
P1 (Session-Replay-Probing). The BotProbe monitor may spoof the address of the server and insert additional TCP packets that replay the same application command P_c to

[5] Sometimes there is no such a message response packet P_r, but rather a activity response. We still use P_r to stand for this activity response.

[6] The false positive rate can be higher particularly if P_r is only a message response packet because it could be just a normal prompt chatting message from a human.



(a) P0,P1,P2,P4: Active probing by injecting packets in existing connection.



(b) P3: Active probing by establishing new connection.

Figure 28: Example active probing techniques. Here cmd' means a modified command packet, seq' means modification is needed on the sequence/acknowledge number to keep the TCP session.

the client several times. If the remote end point is a bot it is likely provide responses that are deterministic (with respect to both content and timing).

P2 (Session-Byte-Probing). The BotProbe monitor may randomly permute certain bytes of the application command.[7] If the client is a bot, then we expect it to be highly sensitive to modifications of commands and hence to respond differently or drop the modified packet. However, a human user in an IRC chatting channel would have a higher tolerance for typographical mistakes in an IRC message. We may repeat our test as many times as necessary by interleaving strategies P1 and P2, until we have sufficient evidence to validate

[7]Since many common botnet command names (e.g, `.dos`, `.scan`) are embedded in the initial bytes of IRC `PRIVMSG` or `TOPIC` message packets, we recommend biasing the byte modification algorithm to choose the early bytes with higher probability.

our hypothesis. We describe the algorithm (Interleaved-Binary-Response-Hypothesis) in more detail in Section6.2.3.

Note that strategies P1and P2 may break existing connections (by injecting new packets) if subsequent C&C communications occur in the same TCP connection. To recover from this, our in-line botnet probing system should adjust the TCP sequence/acknowledge numbers and checksums to account for the new packets that were introduced because of the probes. Also, the above two probing strategies introduce some amount of interference into existing sessions at the application level. Fortunately, we find that, for our targeted chatting-like protocols, we have an alternate probing technique (P3), which does not disturb existing sessions.

P3 (Client-Replay-Probing). Chat protocols like IRC and IM allow users to directly message each other. In such instances, we can instantiate a new user that logs into the channel and sends the observed command(s) P_c to the selected client (pretending to be the botmaster). By doing this, we do not break existing connections, but may achieve an effect similar to that above. Figure 28(b) illustrates this scenario.

P4 (Man-In-The-Middle-Probing). The above techniques do not directly intercept a new command packet. However, in some cases (as discussed in Section 6.4) such as highly stateful C&Cs where simple replaying may not work, we can intercept the *new* command, and launch a man-in-the-middle-like chatting message injection.

P5 (Multiclient-Probing). The above techniques discuss probing sessions from a single client. However, when there are multiple likely-infected clients in the monitored network that are communicating with the same C&C server, we can distribute the probes into multiple clients, and reduce the number of probing rounds we need to test our hypothesis, as briefly discussed in Section 6.4.

6.2.3 Algorithm Design for Botnet Detection Using Active Probing

Based on the active probe techniques, we now describe several simple detection algorithms for isolating deterministic botnet communication patterns from human chatting dialog with controlled accuracy (i.e., to achieve a desired false positive/negative rate). We will use a sequential probability ratio testing (SPRT [108]) technique, which has been applied in several other scenarios such as port scan detection [57] and BotSniffer [48]. To illustrate the algorithm, we start with a basic description of how to apply a hypothesis testing framework using our six probing strategies (P0-P5). We anticipate that all the probing strategies can be iterated through the following testing strategy.

Let us assume that we are given a (suspicious) IRC session and we want to differentiate whether it is more likely a botnet C&C channel or a human chatting session. We perform one or more rounds of P0 probing (i.e., inject a challenge to the client, ask the local participant (within our network boundary) to solve a puzzle). We denote H_1 as the hypothesis "botnet C&C," H_0 as the hypothesis "normal chat." Let a binary random variable D denote whether we observe a *wrong* reply for a challenge from the client or not (that is, $D = 1$ means an incorrect reply). We also denote $\theta_1 = Pr(D = 1|H_1), \theta_0 = Pr(D = 1|H_0)$. If the client is a bot, we presume $\theta_1 \approx 1$, assuming that bots are unable to reliably solve arbitrary puzzles on demand. For a human, such a puzzle is easy to answer, i.e., $\theta_0 \approx 0$. If we want to have very high accuracy for the hypothesis (let us denote α, β as the false positive rate and false negative rate we want to achieve), we can perform several rounds of probing. Then, after observing n rounds, we get a likelihood ratio $\Lambda_n = \frac{Pr(D_1,...,D_n|H_1)}{Pr(D_1,...,D_n|H_0)}$. D_i represents our independent identical distribution (i.i.d.) observation result from our client probe test. We define $\Lambda_n = \ln \frac{\prod_i Pr(D_i|H_1)}{\prod_i Pr(D_i|H_0)} = \sum_i \ln \frac{Pr(D_i|H_1)}{Pr(D_i|H_0)}$. To calculate this likelihood Λ_n, we are essentially performing a threshold random walk. The walk starts from origin (0), goes up with step length $ln\frac{\theta_1}{\theta_0}$ when $D_i = 1$, and goes down with step length $\ln \frac{1-\theta_1}{1-\theta_0}$ when $D_i = 0$. If Λ_n is greater than a threshold $t_1 = \ln \frac{1-\beta}{\alpha}$ we declare H_1 to be true, *i.e.*, it is a botnet C&C. If Λ_n is less than another threshold $t_2 = \ln \frac{\beta}{1-\alpha}$, this indicates a normal

IRC dialog. If Λ_n is in between t_1 and t_2 we proceed with additional rounds of testing. A nice property of this SPRT/TRW algorithm is that it can achieve bounded false positive and false negative rates as desired, and it usually needs only a few rounds to reach a decision [108]. We call our first extension of the algorithm **Turing-Test-Hypothesis** because it uses explicit challenge response. This algorithm even does not require observing any actual botnet C&C interaction.

Similarly, we can adapt the algorithm to use the P1 technique in every round. Let P_c be a suspicious command packet from the server to the client. We replay P_c in each round and we denote D to indicate whether or not a response from the client is observed. We call this **Single-Binary-Response-Hypothesis** algorithm because this test considers the probe response as a binary outcome. Depending on the response we observe (IRC `PRIVMSG` message, scanning, spamming, or third-party access), we iterate the TRW process at different scales, because θ_0, θ_1 (the corresponding probability associated with a bot or human) is different for different responses. For example, a human-driven IRC session is very unlikely to perform scanning when receiving a chatting message. Thus, we improve our confidence when we observe a scanning response corresponding to the replayed (command) message. If we receive multiple different types of responses corresponding to the same command, we choose the one that provides greater confidence (walks a larger step). The exact number of rounds we need in this case is discussed in the next section. In general, Single-Binary-Response-Hypothesis is very effective if the replayed command packet is scan,spam or binary download related. As shown in Section 6.2.4, we may need only *one* extra replay in addition to the original command, i.e., totally two rounds to detect a botnet.

In addition to performing binary response testing, we can further evaluate whether the response is *similar* to the previous response observed, because bot responses may not be perfectly identical across multiple command replays. We hypothesize that for bot C&C communication, responses to the same command will be the same, or very similar (in structure and content). We can design a new hypothesis algorithm that inspects whether a

response is correlated to previous responses using a simple edit distance metric or a DICE metric as in [48]. We call this extension **Correlation-Response-Hypothesis** algorithm.

Finally, we introduce **Interleaved-Binary-Response-Hypothesis** algorithm. In each round, we perform interleaved P1 and P2 probing, i.e., replaying the original P_c packet, and then replaying a modified P_c packet. $D = 1$ denotes the observation of a response from the replayed P_c, and no response from modified P_c. The assertion is that bots will reliably respond to P_c, but will not recognize the modified command. This occurrence is then observed as $D = 1$. To a human user, these two are similar (a modified P_c is just like a typographical error (typo), and in chatting, a typo is normal and generally not a problem). It is hard to predict how normal users may respond when they receive these two replayed IRC `PRIVMSG` messages, but the probability of obtaining repeatable responses from replayed P_c and no responses from modified P_c should diminish with rounds. A naive assumption is that the human responses to tampered packets are uniformly random, $\theta_0 = Pr(D = 1|H_0) = 1/4$. In reality, normal users would quickly lose patience upon receiving multiple similar IRC messages and hence this probability θ_0 should be lower than the uniformly random case. Our later user study (in Section 6.3.3) also confirms that θ_0 is very low.

One benefit of the Interleaved-Binary-Response-Hypothesis algorithm is that we can have a *general* way to detect a *third-party access* response and do not rely on content signatures (e.g., PE signature as used in BotHunter [46] to detect egg downloading). This has the advantage when we do not have signatures for detecting these third-party access, e.g., the access is not for a PE executable, or the access connection does not yield a successful download of a PE executable. We begin by building a suspicious access set containing addresses (most likely, HTTP addresses) that appear after the P_c but not after the modified P_c. Then for each subsequent round, we assign $D = 1$ if we see an address from the suspicious set still appear upon replay of P_c, but not upon sending of the modified P_c.

We have introduced several different detection algorithms. Now we discuss the typical

selection of proper algorithms in practice when facing different type of response or different combination of responses. We think that for a normal host in chatting, the probability of performing a certain (malicious) activity response (e.g., scan, spam) is lower than performing a message response. The general principle we need to follow here is to choose the algorithm that favors the response with the lowest probability and thereby makes the fewest probing and the largest walk in the thresholded random walk. In the following analysis we assume $Prob(scan) \approx Prob(spam) < Prob(3rd - party - access) < Prob(message)$ in the case of a normal chatting client.

If we observe a scan/spam response associated with a command (there might be other responses such as an IRC `PRIVMSG` message), we choose *Single-Binary-Response-Hypothesis* algorithm on the scan/spam response, and ignore other responses. Usually, we only need another active probing (using P1) to declare a botnet as shown in Section 6.2.4 and 6.3.2. It is possible that these scan/spam responses are long-lasting, i.e., we still observe the response to the original command after we perform P1 (a replayed command). However, we do not consider this as a problem, because in any way we still detect the bot. Here our detection performance is at least no worse than the approaches that issue alerts when observing the combination of IRC events and scan events such as [18] and BotHunter [46].

If we observe a third-party access (by matching a PE signature) associated with a command (there might be some message response, but no scan/spam responses), we choose *Single-Binary-Response-Hypothesis* algorithm on the third-party access response.

For the rest combination of responses (e.g., a message response and a third-party access response without PE signature capturing) or only a message response, we can choose *Interleaved-Binary-Response-Hypothesis* algorithm. If there are both a message response and a third-party access observed, to make a walk in the algorithm, we always pick the type of response that can make a larger step (third-party access in this case).

6.2.4 Evaluating User Disturbance and Detection Accuracy Tradeoff

We now describe how the above algorithms can be adapted to trade off user disturbance with system performance. For benign IRC chat sessions, replaying or modifying some byte is essentially equivalent to receiving a duplicate message or receiving a message with a typo: humans have natural resilience to at least limited occurrences of these events. The Client-Replay-Probing technique, which establishes a new session, is even less harmful. Nevertheless, we acknowledge that active modifications to user IRC sessions may impose some degree of cost to human users. We leave a more detailed discussion on the legal concerns of using active techniques in Section 6.4.

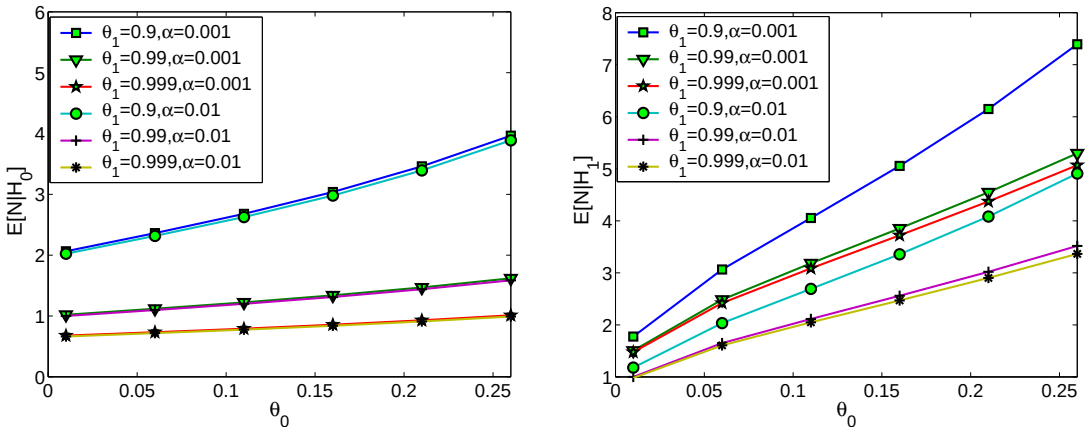
As discussed earlier, in order to have a high confidence of hypothesis testing, we may need N rounds of probing. If we are concerned about the disturbance/interference to normal users, we could use the number of rounds (packets modified/replayed) by active probing as a means to quantify the degree of disturbance. Clearly, less disturbance means fewer rounds, smaller N, which on the other hand, may affect the performance of detection. Fortunately, because of the use of SPRT, the average number of N to make a decision is quite small. To produce a botnet C&C declaration, the expected number of rounds we need is [108]

$$E[N|H_1] = \frac{\beta \ln \frac{\beta}{1-\alpha} + (1-\beta) \ln \frac{1-\beta}{\alpha}}{\theta_1 \ln \frac{\theta_1}{\theta_0} + (1-\theta_1) \ln \frac{1-\theta_1}{1-\theta_0}}$$

Similarly, to produce a human user IRC channel declaration, the expected number of rounds is

$$E[N|H_0] = \frac{(1-\alpha) \ln \frac{\beta}{1-\alpha} + \alpha \ln \frac{1-\beta}{\alpha}}{\theta_0 \ln \frac{\theta_1}{\theta_0} + (1-\theta_0) \ln \frac{1-\theta_1}{1-\theta_0}}$$

Figure 29 shows the average number of rounds we need to declare a normal user (a) or bot (b). For example, if we set parameters $\theta_1 = 0.99$, $\theta_0 = 0.15$, and our desired false positive/false negative rates are $\alpha = 0.001, \beta = 0.01$, then the average number of rounds to declare a botnet is about $N_1 = 3.7$. Likewise, the average number of rounds to declare a human user is less than 2 for IRC (approximately $N_0 = 1.3$). If we observe some scan



(a) Average number of rounds to detect normal IRC user.

(b) Average number of rounds to detect botnet C&C.

Figure 29: Disturbance to normal user and the effect on detection.

response, we can use a lower probability of θ_0, e.g., $\theta_0^{scan} = 0.01$, then it will take less than two rounds (i.g., one extra replay) to detect bots on average.

In practice, our system could be bolstered by an IRC channel whitelist to minimize user disturbance (i.e., once an IRC server/channel is validated, we will not disturb other users for a certain time window, and the time window could be randomized). Finally, the BotProbe strategy should be viewed as one input among a broader set of threat indicators that can be applied for detecting internal botnet infections. For example, the results produced by the BotProbe hypothesis testing framework could be incorporated into systems such as BotHunter [46], which considers the full set of potential botnet-related infection indicators, such as exploit usage, egg download events, and inbound and outbound attack behavior.

6.3 Experiments with BotProbe

6.3.1 BotProbe: a Prototype Active Botnet Probing System

We have implemented a prototype middlebox system called BotProbe for the purpose of evaluating our active probing techniques. BotProbe is implemented as a collection of Click routing elements [60]. Click provides a C++ software framework for packet processing, with impressive scaling performance and a flexible configuration language, that makes it

ideal for building software routers and middleboxes.

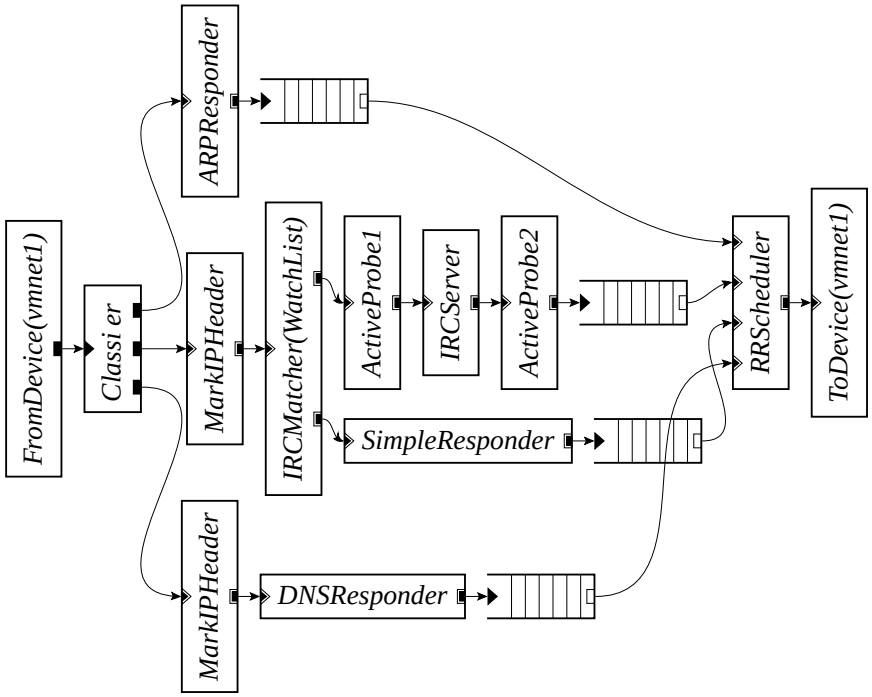


Figure 30: Click configuration for BotProbe. The figure shows a configuration for black-box testing on existing bot binaries. If BotProbe is deployed as a middlebox into a real network, we can remove the IRC Server, SimpleResponder, and DNSResponder elements.

The architecture shown in Figure 30 is implemented in 2,500 lines of C/C++ code. The key elements in Figure 30 that we developed are WatchList, IRCMatcher, and ActiveProbe. WatchList is a Click "information" element that keeps track of live TCP flows and IRC records. The IRCMatcher uses a WatchList to maintains flow records and examines incoming packets to identify IRC flows. The ActiveProbe element monitors all IRC flows, performs active probing if the IRC channel is deemed suspicious, and modifies TCP sequence/acknowledge numbers and checksums when necessary.

To simplify black-box testing on existing bot binaries, we also implemented the following elements: (i) an IRCServer element, which plays the role of a simple IRC server, (ii) a SimpleResponder that handles all non-IRC connections by acknowledging every packet it receives, and (iii) a DNSResponder that answers DNS queries with a local address. If BotProbe is deployed in-line as a middlebox into a real network, we can simply remove

these three elements.

6.3.2 In Situ Experimental Evaluation

We evaluate the detection performance in a virtual network environment with several malicious IRC bots including Sdbot, Phatbot, Rbot, RxBot, Agobot, Wargbot, and IRCBot that we obtained from our bot source code repository and honeynet capture in the wild. The purpose is to test the false negative rate, i.e., how many bot C&Cs are missed by BotProbe? We answer this question using *in situ* VMware testing of real-world bot binaries described below. We also need to test the false positive rate, i.e., how frequently could normal chatting sessions be mislabeled as botnet C&C using BotProbe techniques. We explore this through a user study described in Section 6.3.3.

6.3.2.1 Detection Performance and Analysis

We begin our analysis by conducting a series of *in situ* experiments to evaluate the false negative rate. We proceed by executing the bot in a Windows XP (VMware guest) instance and monitoring with BotProbe running on the Linux host machine. Initially, BotProbe essentially acts as a faithful NAT middlebox interposing all communications between the infected host and the Internet. If the IRCMatcher element identifies an IRC session, the flow will be forwarded to the IRCServer element that handles and responds to all IRC requests. The ActiveProbe element resides between the bot client and the IRCServer element, monitoring chatter and introducing active probes at appropriate time (e.g., when the channel is idle on a suspicious session). While the IRCServer element has the actual botnet commands, we do not assume the ActiveProbe element knows the commands, as BotProbe runs in the realistic scenario.

Note, in real-world IRC based botnets, we observe most of the commands are in IRC `TOPIC` messages. This is because that botmasters are not online all the time. In order to instruct bots even when they are not online, botmasters usually put the commands in the `TOPIC` of the channel. Thus, whenever a bot joins the channel, it will understand the

commands in `TOPIC` and execute (without authentication). In such cases where there is no `PRIVMSG` message from the server but client responses are still produced, we can presume the `TOPIC` is the command and play the probing game by manipulating observed `TOPIC` messages (332). We use this trick in our experiments, in order to faithfully replicate real-world scenario. In addition, as discussed earlier in Section 6.2.3, BotProbe performs Single-Binary-Response-Hypothesis or Interleaved-Binary-Response-Hypothesis algorithm in our experiments depending on what kind of (combination of) response(s) it observes.

We evaluate BotProbe on several real-world IRC bots that can be grouped into three classes.

1. Open-source bots with obfuscated communication. Our first case study is an "open source" (as described in the bot documentation) IRC bot called Spybot, which was released in 2003. Being open source, many variants of this bot are on the Internet, making it one of the more popular botnet families [16, 40]. Spybot is also equipped with a command encryption option that obfuscates C&C communication. The encryption method implemented is a simple byte shift scheme. We recompiled the Spybot source with the encrypt option enabled and tested the binary using BotProbe.

In evaluation, we configured the IRCServer to issue a set of commonly used commands listed in Table 12 (one command in each test). We set the parameters of the hypothesis testing algorithm to be $\theta_1 = 0.99, \theta_0 = 0.15$ giving expected false positive (FP) and false negative (FN) rates of 0.001 and 0.01, respectively.[8] We set $\theta_0^{scan} = 0.01$, because the probability that a normal chatting client has scan response is low (much lower than an IRC message response). Similarly, for a third-party access response, we set $\theta_0^{3rd-party-access} = 0.02$. We used this parameter setting in our entire experiments. In the test on Spybot, BotProbe

[8]This θ_0 is for the case of Interleaved-Binary-Response-Hypothesis on message response only.

took two probes when the command was “scan” (Single-Binary-Response-Hypothesis algorithm was automatically performed), two probes when the command was “download” (Interleaved-Binary-Response-Hypothesis algorithm was automatically performed because we do not use any PE signature to identify access response), and four probes when using commands such as “info” and “passwords” (Interleaved-Binary-Response-Hypothesis algorithm was automatically performed).

Table 12: BotProbe test on Spybot and Rbot.

Bot	Original cmd	Obfuscated cmd	IRC response	Activity response	No. rounds
Spybot	`info`	`otlu`	`Version:... cpu:...`	-	4
	`passwords`	`vgyy}uxjy`	`Error operation failed`	-	4
	`scan 192...`	`yigt&7?847<>477<4<&79?&7`	`Portscanner startip:...`	scan	2
	`download http:...`	`ju}trugj&nzzv@55oxi...`	`download http:...`	3rd-party access	2
Rbot	`.id`	-	`[MAIN]: Bot ID: rx01.`	-	4
	`.sysinfo`	-	`[SYSINFO]: [CPU]: ...`	-	4
	`.scan 192...`	-	`[SCAN]: Port scan started...`	scan	2

2. Bot binaries with clear-text communication. We tested a few other bots, e.g., Phatbot, Rbot, Rxbot, Sdbot [16], in our controlled network. In these experiments, C&C exchanges are in clear text by default. However, we noticed that the source code for these bots includes encryption and decryption functions, shell code encodings, and support for polymorphism. It is straightforward to enable the use of these encryption routines for command obfuscation. The performance of BotProbe on these bots was identical to Spybot, i.e., it took two or four rounds of probes, depending on the command.

3. Bot binaries with obfuscated communication. Next, we tested on a recent bot binary (W32.Wargbot as labeled by Symantec) captured in the wild [28]. The botmaster put an encrypted command (shown below) in the IRC `TOPIC` message for bots to execute upon join. Subsequently, BotProbe automatically performed Single-Binary-Response-Hypothesis algorithm, and it took only one extra probe to declare the bot because the bot had background scanning behavior.

```
!Q ;\\|!Q <W:Z<Z=B=B=>;P;E;E<[=;<Y=>=:<S<U<W;D<U===;;E<V<[=@<W<U=B===@=G;E<V
<[=@;I;O;N;O;E;G;G;N;G;J;H;L;G;O;H<Q;M;J;F;K;D<\\=><Y
```

We then enabled Interleaved-Binary-Response-Hypothesis algorithm on the same bot. Again BotProbe took two total rounds to declare the bot, and this times it reported observing a third-party access response (the bot initiated an HTTP GET request when it received the `TOPIC` command), which was suppressed in the previous test because BotProbe automatically chose Single-Binary-Response-Hypothesis once it observed scanning behavior (which yielded a larger walk in TRW than the case observing a third-party access response, as discussed in Section 6.2.3). This third-party access response is interesting because it established some mapping between the obfuscated command and the corresponding visited URL. By intentionally changing different portion of the obfuscated command and watching the corresponding URL, we can perform a chosen-ciphertext attack to crack the obfuscation scheme. BotProbe again demonstrated its extra utility in automating the informed active probing and collecting the mapping for our analysis, in addition to detecting the bot. By interrogating the bot with single byte modifications using BotProbe we were able to reverse engineer the encoding scheme used by the bot. The actual command after decoding is

```
" F |" e http://img2.freeimagehosting.net/uploads/03bd27490b.jpg
```

Here the original obfuscated `!Q␣` (`"␣`) is likely to be a command prefix and `|` is a separator between commands. We are unsure about the meaning of the translated "`F`" command, but suspect that "`e`" is a download command followed by a URL. Breaking the encoding/decoding scheme is interesting because it enables us to decode other commands we observe for different variants of this botnet. In our honeynet, we have observed at least two other commands issued by the same botnet (with different bot binaries).[9] The changes in commands reflected relocation of binary hosting websites and file names. Apparently, the original hosting site (`media.pixpond.com`) was no longer available, so the botmaster switched to two other websites (`imgplace.com` and `img2.freeimagehosting.net`).

[9]We know that it is the same botnet because the binaries use the same C&C channel.

Although we successfully launched a chosen-ciphertext attack to break the encryption scheme of some botnets with the assistance of BotProbe, there are still some cases where we could not break the scheme. However, these instances were all successfully detected as botnet C&C by BotProbe. In June 2007, we captured a bot in the SRI honeynet, which is labeled as Trojan.Dropper.Sramler.C by several AV tools. This bot uses C&C obfuscation and it makes several DNS requests, which all translated to the same IP address, demonstrating multiple degrees of stealthiness. The command we observed (shown below)[10] is apparently an update/download command because BotProbe successfully identified a third-party access response (using Interleaved-Binary-Response-Hypothesis algorithm and probing only two rounds), i.e., a download from `http://220.196.X.107/packed_7711.exe`.

```
=xAgVMf81RvN+xBBhG+xXwttpTsaSBfWeekvMkmkVNcbo20jZvmkCo7CUUbRsdRPzz6wiS1O
Y8pcXg3d9ucVufq2bgQ1mvh+9OBJDwIuw1kOamPaw+2jw/CTaWVQRjrX8Xl2Iph
```

In September 2007, we captured a new variant of this bot, which is labeled as Backdoor.Win32.IRCBot.aby by several AV tools. We verified that this is essentially the same botnet as the aforementioned botnet, as they both contacted the same IRC server 220.196.X.226, The bot observed in June contacted port 3938 while the later bot contacted the server on port 2234 with the following command:[11]

```
=YXCdm8MDxhmOoBo3aSrxyp83pM5yZRnQVt8O+mVxm9bwLd77Ahc6KWKVn/DWu+ACn4mrpT
j6U5+yXie37WfPaymQmLtbkxPUVB2JaMwddAVokDxqsbjxmPlqpjeQIh
```

It turns out that this is actually an access to `220.196.X.107/kk.exe`, and BotProbe took only two rounds to flag this as a botnet C&C communication. To conclude, BotProbe has a 100% detection rate in recognizing IRC-based botnet C&Cs, despite the presence of obfuscated communication.

[10] At a first glance, this looks like a BASE64 encoded string. However, we verified that this is not the case, at least not a pure BASE64 scheme.

[11] The fact that the same IP address remained as the C&C server for over 3 months suggests that obfuscated botnets might be more resilient to detection.

6.3.3 User Study on Normal Chat Probing

We report on the results of a user study that simulates the impact of active probing techniques on human chat sessions. This study was conducted at the Georgia Institute of Technology.

Study design and ethical guidelines: Since we are not allowed to *directly* alter live network flows on campus, we recruited human users to go online and chat with real users at diverse channels on multiple networks. During the chat sessions, our human users periodically sent crafted messages that simulate the effect of botnet probing. Our goal was to confirm our hypothesis about human response to tampered messages and evaluate the degree to which simulated BotProbe techniques affect normal users, e.g., how many actual rounds would we need on average to detect a normal user? While our current study is limited to two different chat platforms, IRC and `meebo.com` (a website providing instant messaging and chat room capabilities), we believe that our results hold across chat platforms because they simply capture basic human responses.

Our study protocol was reviewed and approved by the institutional review board (IRB) of Georgia Tech. To alleviate any privacy concerns, we anonymized usernames and IP addresses and recorded only the following necessary information: messages exchanged and timestamps. Furthermore, although we introduced additional network flows, our methodology caused no interference to existing IRC network flows.

Participant selection: We logged into different IRC/meebo sites/channels and randomly selected active chat users who were exchanging chat messages in the channel when we logged in. We started engaging them in conversations just like normal users. The users we contacted were not aware of the study or the active probing techniques/algorithms that we employed. This was necessary to ensure the fairness of our testing procedure.

Study procedure: We designed six different question sets to test on 123 different users. Our question set includes simple messages like “what’s up,” “nice weather,” “you like red?”“How may I help you?” “English only! I play nice fun” and Turing test messages

such as “what’s 3+6=?” As we conversed with a user on a chatting channel/room using a random question set, we deliberately introduced probing at certain predefined points. We then measured the user’s responses to these tampered messages. The conversations we recorded could be broken down into two classes.

First, although we randomly chose a user who seemed to be active in the chats room/channel, there is always a chance that the user does not respond to our overtures. Such cases occurred 26 times (no active replies to our messages). We discount these cases from subsequent analysis. Second, if the user was willing to pursue a conversation, by responding to our first question, we followed by sending two or three rounds of repeated questions that interleave original and slightly tampered messages (by introducing a typo in the first few bytes of the message). Some examples of tampered messages include “waat’s up,” “noce weather,” “aou like red?” “Bow may I help you?” “Eaglish only! I play nice fun.” This simulates the behavior of BotProbe performing P1/P2 probing. We recorded the exchanged messages for evaluating Interleaved-Binary-Response-Hypothesis algorithm. In addition to P1/P2 probing, we subjected the user to P0 probing using Turing-Test-Hypothesis algorithm described above.

Table 13: User study of performing P1 and P2 probing, using Interleaved-Binary-Response-Hypothesis algorithm. Most users are detected as normal in two or three rounds.

	meebo chats	**IRC chats**	**Total**
Detected in 2 rounds	63 (75%)	10 (77%)	73 (75.3%)
Detected in 3 rounds	8 (9.5%)	1 (7.7%)	9 (9.3%)
Pending after 3 rounds	13 (15.5%)	2 (15.3%)	15 (15.4%)
Total	84	13	97

User study of Interleaved-Binary-Response-Hypothesis: In total, we tested 97 different users, 84 on meebo and 13 on IRC. A simulated BotProbe can detect most of the normal users (75.3%) in just two rounds and 9.3% in three rounds. The rest (about 15%) are marked still pending. We provide a summary of our results with respective breakdowns for meebo and IRC in Table 13. We set our probing to be three rounds to limit annoyance/interference to chat users. We further believe that most of the pending sessions can be

easily declared as normal users by sending additional probes (we selectively verified this on a few cases). Finally, we did not encounter any false positives (misclassifying a normal user as a bot) in our limited testing.

User study of Turing-Test-Hypothesis: In addition to P1 and P2 probing, we tested P0, i.e., injecting Turing test messages (but without user education). We performed tests on 30 different users in meebo. The basic question/puzzle we sent is "what's 3+6=?" Although all users provided the correct answer upon repeated interrogation, we found it difficult to get a direct answer the first time the question is posed. These users tend not to answer this in a correct way, possibly because they thought it might be unnatural to receive such Turing questions in the chatting channels (they perceive this to be some sort of a joke). We conclude that if users are not educated to be familiar with such Turing tests or have an unusually strong desire to be in a channel, it is difficult to perform generic Turing tests on IRC or meebo networks. This also illustrates that although P0 probing seems simple and effective (if users are educated), there is still a need for alternative and transparent techniques that require no explicit user education (like our P1-P5 techniques).

6.4 Discussion

6.4.1 Legal Concerns

Legitimate "bots": It is likely that in some cases there are legal "bots," e.g., some client-side legitimate programs or automatic scripts that build their application logic over the chatting protocols such as IRC. For instance, some chatting bots [42] can also be detected by BotProbe. A possible solution is to whitelist these legitimate applications if they are very important and critical, and do not want to be disturbed (we expect such applications to be very few). However, we think probing a pure chatting bot is not very critical, and arguably, the detection of such a chatting bot is not considered as a false positive. Furthermore, there are several heuristics that can help differentiate these chatting bots from real malicious bots. For example, unlike malicious bots, chatting bots are not likely to generate activity

responses (e.g., scan response). In addition, we can consider a group property (similar to the group analysis in BotSniffer [48]) to differentiate a malicious botnet, where clients in the same channel are mostly bots, from a normal human chatting channel with mostly human and very few chatting bots.

Legal implications: Our active probe techniques might be deemed controversial because they alter network flows to human users, and may replay malicious commands. In Section 6.2, we have discussed the tradeoff between detection accuracy and disturbance to human users and various means to mitigate interference with legitimate chat sessions. We now consider potential legal implications of replaying a "potentially malicious command packet." First, we argue that to minimize liability issues, the only packets that should be tampered with by BotProbe are packets that are inbound to the local network. Second, this potentially malicious command has already been transmitted into our network and executed on the local host prior to our probing. Third, if the purpose of the command is information gathering (e.g., `.sysinfo`), then we argue that the first command-response already leaks enough information, and our further replay most likely does not leak more information or perform more harm. In short, although controversial, we believe that the benefits of actively probing suspicious sessions could outweigh the potential disturbance.

6.4.2 Limitations and Potential Solutions

As stated in Section 6.1, BotProbe has clear assumptions to limit its application to certain class of botnets that use chatting-like C&C. Next, we describe some possible evasions,[12] though we have not observed real examples yet, and discuss our potential solutions.

Strong encryption: Active probing techniques cannot identify botnet C&C channels that use strong encryption schemes (e.g., SSH, SSL) making them resilient to replay attacks. Note, however, passive perimeter monitoring strategies cannot detect, too, and most contemporary IRC bots avoid or use weak encryption schemes. At a minimum, BotProbe

[12]Most of these evasions are against Single-Binary-Response-Hypothesis and Interleaved-Binary-Response-Hypothesis algorithms. That is, Turing-Test-Hypothesis algorithm could still work.

raises the bar and forces all botnets to adopt strongly encrypted communications. This is also an important area for future work. We envision that combining host-based monitoring could be helpful.

Timer-based evasions: Knowledgeable adversaries could design bots to have programmed timers that greatly delay the response time, or limit the number of commands of the same type that could be issued to the bot in a certain time window. By using such timers, a bot can potentially evade our Single-Binary-Response-Hypothesis algorithm. Note, however, that this would also reduce the efficiency of the botnet because the botmaster cannot command the botnet promptly, or to repeat the same task for a certain time. Our potential solution against such an attack is to randomize the delay in command replays.

Stateful C&C protocols: Our P1 and P2 probing techniques assume a stateless C&C protocol, i.e., we can replay the observed command several times, and the bot always responds similarly to the same command. In the future, botmasters may create a stateful command processor that can detect duplicate commands, e.g., by using a timestamp or sequence numbers with every command sent, making simple replay ineffective. Note, most contemporary IRC botnet command-response protocols are stateless and deterministic. In addition, our P0 probing can still work even in this evasion. Moreover, to counter this possible future evasion, we describe a potential solution if there are multiple command-response rounds and multiple clients in the monitored network. Instead of replaying packets, we could intercept and modify chatting packets sent to subsequent clients by using P4 and P5 probing techniques. By intentionally modifying the command sent to some clients while leaving commands to other clients untouched, we could measure the difference in response between messages, which would be analogous to replaying the command to the same client several times in an Interleaved-Binary-Response-Hypothesis test.[13]

Finally we remark that future work is definitely needed in this area. While imperfect and clearly limited, active techniques such as BotProbe can greatly complement existing

[13]Here we assume the C&C is one-to-many, i.e., one command to many clients in the network.

passive detection techniques, at least for a large portion of real-world botnets.

6.5 Summary

In this chapter, we propose the idea of using active probing techniques to detect botnet C&C communications that use chatting-like protocols. By requiring the observation of *at most one* round of actual C&C interaction and then applying active probing, this approach, unlike existing passive approaches, can actively collect evidence and shorten the detection time. We have developed a hypothesis testing framework and a prototype system implementation that effectively separates deterministic botnet communication from human conversations, while providing control over false positives and detection rates. We validate our system on several malicious IRC bots and conduct an actual user study on approximately 100 users. Our experimental results, while preliminary, are encouraging. BotProbe is not intended to replace existing passive detection approaches, but to complement them from a new perspective.

This work represents the first feasibility study of the use of active techniques in botnet research; thus, we hope to inspire new thought and directions in the research community. While controversial and clearly limited, BotProbe has demonstrated its effectiveness in detecting a large portion of contemporary real-world botnets. In future work, we will study robust, practical, and less controversial extensions of active techniques for a more general class of botnet C&C detection. In addition to detection, active techniques can be used for other purposes, e.g., probing the server side, and injecting watermarks to trace the location of a botmaster. We plan to investigate these new potential utilities of active techniques in the future.

CHAPTER VII

LESSONS LEARNED AND A FUTURE BOTNET DETECTION SYSTEM

We have introduced four Bot* systems: BotHunter, BotSniffer, BotMiner, and BotProbe. In this chapter, we plan to answer the following questions: How are Bot* systems related to and different from each other? What lessons have we learned from designing these systems? How can we design a future botnet detection system that combines multiple techniques we have used in the Bot* systems?

7.1 Summary of Bot Systems*

Table 14 provides a brief summary of our Bot* systems under the taxonomic study we proposed in Chapter 2.

Table 14: Summary of our Bot* detection systems.

	BotHunter	BotSniffer	BotMiner	BotProbe
Host or network based?	network	network	network	network
Signature or behavior based?	behavior	behavior	behavior	behavior
Passive or active?	passive	passive	passive	active
Detection phase	preparation, or operation	operation	operation	operation
Detection target	individual	group	group	individual
Require out-of-band information?	no	no	no	no
Restriction on botnet C&C technique	general	centralized structure	general	chatting-like C&C
Correlation technique	vertical (dialog)	horizontal	horizontal	cause-effect
Offline or online correlation?	both	both	offline	online

These systems have similar features. A common thread in all these systems is the use of correlation analysis, i.e., vertical (dialog) correlation in BotHunter, horizontal correlation in BotSniffer and BotMiner, and cause-effect correlation in BotProbe. In addition, all Bot* systems are network-based and behavior-based approaches, and they do not require out-of-band information.

Despite the above similarities, these systems differ in several dimensions. Unlike the

other three systems, which use a passive monitoring strategy, BotProbe uses an active approach. BotHunter can detect a successful bot infection in either its preparation phase or its operation phase, while the other three systems can detect bots in only their operation phase. BotHunter and BotSniffer detect bots from their *individual* behavior, while BotSniffer and BotMiner detect botnets from their *group* behavior. Regarding restrictions on botnet C&C techniques, BotSniffer focuses on the effective detection of centralized Botnet C&C channels, and BotProbe focuses on the fast detection of Botnets that use chatting-like C&C protocols. BotHunter is independent of botnet C&C techniques, though it depends on the bot infection dialog, and its current C&C sensor is mainly signature-based. BotMiner is truly independent of botnet C&C techniques among the Bot* systems.

Each of these systems has its own advantages and disadvantages. BotHunter is good at detecting bots that follow an infection model consisting of several infection stages, and it can potentially issue alerts in the early phase of the bot infection before bots are fully controlled to perform further malicious activities. However, it is restricted to the predefined infection model (although this model is relatively general and unlikely to change dramatically, and we also provide an open and flexible framework for easy extension) and at some stages such as C&C communication, it currently provides only signature-based sensors. BotSniffer does not necessarily require the observation of multiple *different* stages on an individual host, and it does not require botnet-specific signatures. Moreover, it can detect botnets within a reasonable time (quicker than BotMiner) as long as multiple instances/rounds of botnet communications/activities are observed. However, it is limited to the detection of botnets mainly using centralized C&C channels. BotMiner overcomes the weakness of relying on a certain botnet C&C technique by providing a general protocol- and structure-independent botnet detection scheme. Hence, it can resist the modification and evolution of botnet C&C techniques. However, its correlation analysis is offline and slow because current clustering features on C-flows are based on mainly statistical distribution within a long time window (e.g., several hours), thereby taking a relatively longer time

for correlation and detection. BotProbe overcomes the weakness of the passive approach, which usually requires the observation of multiple instances/rounds/stages of botnet communications/activities. It shortens the detection time by requiring the observation of *at most* one round of actual botnet C&C interaction. Furthermore, unlike BotHunter's C&C sensor, it does not use a signature-based approach for recognizing C&C. However, it is limited to a certain type of C&C that uses chatting-like protocols. Nevertheless, this weakness can be complemented by the other three systems. As we can see, although these systems have their advantages and disadvantages, they greatly complement each other by being united, and provide a relatively comprehensive and multi-perspective correlation-based framework for botnet detection.

7.2 Lessons Learned

As we have highlighted earlier regarding the challenges in botnet detection, botnet is so far the most advanced and flexible malware form, evading detection that uses simple symptoms or single perspective. From the failure of previous detection approaches (as discussed in Chapter 2) and the success of our designed systems in their desired detection scope, we have learned the following important lessons.

Evidence-trail collection and dialog-based correlation analysis can be effective for detecting advanced malware infections that incorporate multiple stages. Malware can evolve quickly in forms, becoming more and more complex and advanced. However, at the same time, successful malware infections involve more and more stages. For example, compared with previous simple infections, a bot infection typically also involves egg downloading (to have a rich-functional binary) and C&C communications (to be fully controlled). In addition, in the operation phase, a bot typically participates in multiple tasks. An evidence-trail- and dialog-based correlation approach such as BotHunter can effectively collect sufficient evidence to detect such kind of advanced malware. This general "dialog correlation" principle could be applicable to future malware.

Horizontal correlation analysis can be effective for detecting distributed and coordinated malware infections/attacks. A trend of malware attacks is to become distributed and coordinated so that the attacks can be more robust and powerful. Horizontal correlation is an effective way to achieve a big picture of such attacks by correlating the similarities across multiple hosts. Sometimes, it is possible that by using a horizontal view, one can identify malware infections/attacks that are otherwise hardly noticeable at an individual host. We believe this general "horizontal correlation" principle could be applicable to detecting future malware that incorporates distributed and coordinated behavior.

Active techniques can greatly complement passive techniques, but they should be carefully used. As we have shown, compared with passive approaches, an active strategy has unique advantages such as shortening the detection time. These advantages make it very effective for detecting a large portion of real-world IRC-based botnets. We believe active techniques can have broader applicable areas in future malware detection and defense. However, such active techniques should be carefully designed because of their controversial nature, and sometimes they should be coupled with user education and awareness.

An effective botnet detection solution should capture some invariants (rather than symptoms) of botnets, design detection sensors/techniques across multiple events (or stages, aspects), and then combine them for a collective/correlative view. Each of our designed Bot* systems follows this principle, and each system correlates multiple events/stages/aspects for a final detection decision. This principle/lesson can be applicable to situations across the Bot* systems. We know that botnet detection is very difficult and, at present, no *single* technique that perfectly detects *all* botnets is available. Even our designed Bot* systems are effective only within their own *desired or defined detection scope*. However, we can combine multiple techniques to cover multiple perspectives so as to improve the detection coverage of botnets, similar to the case in which we combine multiple perspectives in designing each single Bot* system. This thesis provides our experience in designing different techniques and systems for botnet detection from a certain perspective.

We have also explained why our Bot* systems are different and complementary. In the next section, we discuss how these techniques may be combined to create a future botnet detection system.

7.3 Combining Multiple Techniques in a Future Botnet Detection System

As we have discussed before, our Bot* systems complement each other very well. Figure 31 shows the architecture of an example design of a future botnet detection system that incorporates the complementary techniques we have discussed.

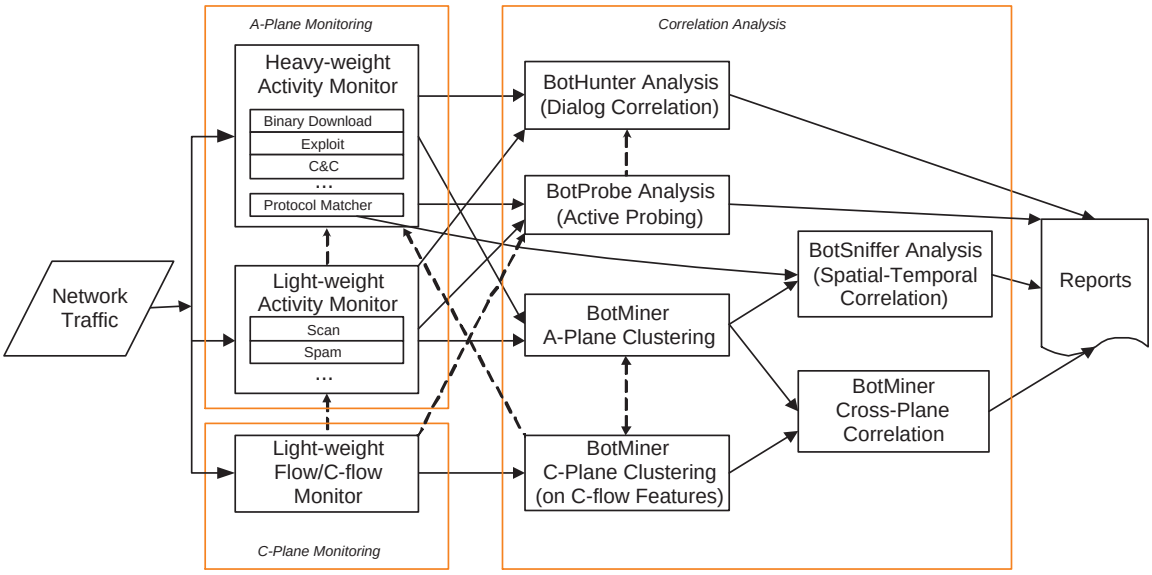


Figure 31: Example combination of multiple techniques in a future botnet detection system.

The new system is divided into two main parts: monitoring components and correlation analysis components, similar to the general design principle of the Bot* systems. The monitoring and correlation components cover both the C-plane (communication plane) and the A-plane (activity plane). The role of (online) monitoring components is to monitor real-time network traffic and generate flow/activity logs that capture "who is talking to whom" (C-plane monitoring) and "who is doing what" (A-plane monitoring). These logs are then further analyzed by correlation components (either online or offline) to identify "who the

bots are within the same botnet because they share both similar communication patterns and similar activity patterns (BotMiner analysis), or because they are doing similar and synchronized activities multiple times (BotSniffer analysis)," and "who a bot is because it has performed a sequence of activities that follow an infection dialog model (BotHunter analysis), or because it has a command-response C&C pattern with strong cause-effect correlation (BotProbe analysis)."

The C-plane monitor is the same component as in BotMiner. It is usually a light-weight network monitor that simply captures flow-level information for recording "who is talking to whom," and our C-flow statistic features will further capture "talking in what kind of patterns."

The A-plane monitor, for capturing "who is doing what," is further divided into heavy-weight and light-weight components according to whether performing deep packet inspection or not. The light-weight A-plane monitoring components mainly look at the packet header information. Scan detection (e.g., SCADE) is a typical example of such a light-weight activity monitor component because it only needs to track information such as the number of (failed) connections. Spam detection is another example that looks for the number of SMTP (port 25) connections and the number of DNS MX queries. In contrast, heavy-weight A-plane activity monitoring components usually need to inspect the deep payload to find specific fields or patterns, e.g., to detect PE binary downloading, exploit, or C&C. They can be based on signature (e.g., rule sets in BotHunter) or anomaly (e.g., SLADE). In particular, we include protocol matchers to locate specific interested protocols such as IRC/HTTP in a port-independent way, and this protocol recognition can be further used by BotSniffer or BotProbe analysis.

After the monitoring stage, flow logs and activity logs are processed by various correlation engines in parallel. Flow records are converted to C-flow statistical feature records and further clustered by the BotMiner C-plane clustering component. Similarly, activity logs are clustered. Then, BotMiner cross-plane correlator will examine groups of hosts

that share both similar communication patterns and similar activity patterns. The BotMiner correlation analysis is usually offline mainly because the conversion of C-flow features requires statistical distribution information within a relatively long period, and the clustering process on numerous C-flow records is also slow.

To complement the slow processing of botMiner, we also perform BotSniffer spatial-temporal correlation on the activity clustering and use the connection information provided by protocol matchers. Thus, if we observe that groups of hosts have multiple rounds of similar behavior and share a common (IRC/HTTP) server connection, we can reach a conclusion of botnets more quickly than BotMiner (because we do not need to wait for the clustering results of C-plane communication patterns).

The above two correlation analysis components can identify which groups of hosts that are likely bots within a botnet. At the same time, if there is only one bot in the monitored network, we may miss detection. Therefore, we need BotHunter's dialog correlation technique so that even if there is only one bot, we can still declare it by observing its infection dialog. Although this detection is costly because it typically requires deep packet inspection, it can detect a bot more quickly than BotMiner and provide a relatively complete profile on the bot infection dialog.

Furthermore, to overcome the weakness of the longer time monitoring requirement of BotSniffer (which needs to observe multiple rounds/instances of botnet communications/activities), BotMiner (which needs to observe a long time window of botnet communication), and BotHunter (which needs to observe multiple different infection stages), we also perform BotProbe analysis. If we identify suspicious chatting-like communications such as IRC (reported either from an IRC protocol matcher or from a general traffic profile matcher from flow records, as mentioned in Section 4.4), we can further apply an active botnet probing technique that detects the existence of bots quickly. Such behavior-based detection can further contribute to the C&C detection stage for BotHunter and a more complete profile of the bot infection dialog. In addition, BotProbe is the only technique among

the Bot* systems that can detect a bot when only one command-response message interaction occurs in a botnet C&C channel without any activity response.

The dashed lines in Figure 31 depict *possible* workflow. A dashed arrow line connecting two components usually indicates that a certain component can be bootstrapped from another for efficiency purposes. For example, both C-plane clustering and A-plane clustering can be bootstrapped from each other, as mentioned in Section 5.2.3. Bootstrapping C-plane clustering from A-plane is particularly useful because it can greatly reduce the amount of work for clustering by focusing on only a small set of C-flows that involve hosts that have demonstrated malicious activities in the A-plane.

Here, we illustrate another example of using efficient bootstrap to work in a high-speed network in which we may not afford to perform deep packet inspection on *all* traffic. We can start from light-weight monitoring such as flow capturing and scan/spam detection. Hosts reported by light-weight activity monitoring components can be fed into heavy-weight activity monitoring. At the same time, if our C-plane clustering component locates a group of hosts that has very similar communication patterns, we can further ask heavy-weight activity monitor components to examine these hosts.

To conclude, although botnet detection is generally difficult, we can develop a relatively comprehensive solution by combining multiple complementary detection techniques to achieve a multi-perspective view, as shown in this section.

CHAPTER VIII

CONCLUSION AND FUTURE WORK

8.1 Conclusion

Botnets are considered as the largest threat to Internet security today. Millions of computers are compromised on the Internet, and they can be controlled by botmasters to launch Internet-wide attacks and fraudulent activities. Thus, we urgently need solutions for the detection of botnets to further mitigate and defend against them.

In this thesis, we have proposed a correlation-based framework for botnet detection in an enterprise-like network environment. Our framework includes three correlation techniques (vertical/dialog correlation, horizontal correlation, and cause-effect correlation) and four prototype detection systems (BotHunter, BotSniffer, BotMiner, and BotProbe). We have discussed the techniques used in each system, summarized the lessons we have learned, and presented an example architecture for combining these complementary techniques into a future botnet detection systems.

Our botnet detection solution meets the four design goals proposed in Chapter 1:

First, each Bot* system is guided by a sound correlation analysis principle that captures some fundamental invariant of botnet behavior. Vertical correlation (used in BotHunter) captures the dialog nature in the multi-stage bot infection life cycle. Horizontal correlation (used in BotSniffer and BotMiner) captures the coordination and similarity nature within the same botnet. Cause-effect correlation (used in BotProbe) captures the non-human driven, deterministic command-response pattern of certain class of botnet C&C channels. We believe the principles of using correlation analysis can also be applicable to detecting future malware.

Second, our solution provides four complementary techniques and covers multiple

stages, dimensions, and perspectives. BotHunter uses *vertical (dialog) correlation* to examine the behavior of each *distinct* internal host and then recognize a bot infection dialog. BotSniffer and BotMiner use another complementary strategy, *horizontal correlation*, to recognize behavioral similarity and correlation *across multiple hosts*. In particular, BotSniffer focuses on capturing multiple rounds of spatial-temporal correlation behavior to detect *centralized* botnet C&C channels, while BotMiner provides a more *general* framework for protocol- and structure-independent botnet detection. Unlike the above *passive* monitoring strategy, which usually requires a relatively longer detection time, BotProbe uses *active botnet probing* techniques in a middlebox to obtain enough confidence of a *cause-effect correlation* caused by the command-response pattern of botnet C&C, and only requires observing *at most one* round of actual C&C interaction. Each system has advantages and disadvantages, and works well in its desired detection scope. We combine these different correlation strategies and different detection techniques/systems to provide a comprehensive and complementary correlation-based framework for multiple-perspective botnet detection.

Third, our solution is general and extensible. In design, each system is not restricted to a specific botnet *instance*, but instead, it targets a certain *class* of botnets. As a representative example, BotMiner can detect a very general class of botnets using different C&C techniques. Even BotProbe, although looks like specific to IRC botnets, is applicable to a general class of botnets that have deterministic, interactive C&Cs (e.g., chatting-like communications such as IRC or instant message). In addition, all these systems are open and extensible, so adding a new detection sensor to an existing system is quite easy. For example, in Section 3.4, we have shown that new detection modules can easily be incorporated into the BotHunter system.

Finally, our systems are practical and capable to work in the real world. Our systems are evaluated on real-world network traces and/or in live network operations. Experimental results are promising, showing that our systems can accurately detect real-world botnets

with a very low false positive rate on real-world normal network traffic. Our work has already begun to make an impact in the real world. For example, BotHunter is available to the public at `http://www.cyber-ta.org/BotHunter/`, and it has amassed more than 6,000 downloads in the first five months since its public release.

8.2 Future work

In the future, we plan to study the following directions:

- Improvements on the efficiency and robustness of existing components in the Bot* systems. We plan to study new techniques to improve the efficiency and increase the coverage of existing monitoring and correlation components, and such techniques are intended to be more robust against evasion attempts.

- Botnet detection in high-speed and large-scale networks. We plan to develop a new generation of real-time detection systems combining vertical and horizontal correlation techniques seamlessly, using a layered design, a flexible sampling strategy, and a highly scalable distributed scheme, and intending to work in very high-speed (e.g., up to 10G bps) and very large (e.g., up to ISP level) network environments.

- More robust and less controversial active techniques with wider applicable areas. As mentioned before, we plan to investigate the practical utility of active techniques for more areas in botnet research.

- Cooperative detection combining host- and network-based systems. Host-based approaches can provide different information/views that network-based approaches cannot. The combination of these two complementary approaches can potentially provide better detection results, e.g., possibly for detecting a highly evasive botnet that uses strongly encrypted C&C. We plan to develop new host-based approaches and study new cooperative techniques in the future.

- Botnet mitigation and defense. Once we detect bots/botnets, a logical question that follows is how we mitigate, respond to, and defend against them. We plan to investigate effective and efficient techniques for achieving this goal in the future.

REFERENCES

[1] "Agobot (computer worm)." http://en.wikipedia.org/wiki/Agobot.

[2] "A guide to understanding covert channel analysis of trusted systems, version 1." NCSC-TG-030, Library No. S-240,572, National Computer Security Center, November 1993.

[3] "Financial insights evaluates impact of phishing on retail financial institutions worldwide." CRM Today. http://www.crm2day.com/news/crm/EplAlZlEVFjAwhYlkt.php, 2004.

[4] "Malware infections in protected systems." Research Study of PandaLabs, http://research.pandasecurity.com/blogs/images/wp_pb_malware_infections_in_protected_systems.pdf, 2007.

[5] "Hi-performance protocol identification engine." http://hippie.oofle.com/, 2008.

[6] "Internet security threat report - Symantec Corp." http://www.symantec.com/threatreport/, 2008.

[7] "Malicious software (malware): A security threat to the internet economy." http://www.oecd.org/dataoecd/53/34/40724457.pdf, 2008.

[8] "Overnet." http://en.wikipedia.org/wiki/Overnet, 2008.

[9] ABAD, C., TAYLOR, J., SENGUL, C., YURCIK, W., ZHOU, Y., and ROWE, K., "Log correlation for intrusion detection: A proof of concept," in *Proceedings of the 19th Annual Computer Security Applications Conference (ACSAC'03)*, (Washington, DC, USA), p. 255, IEEE Computer Society, 2003.

[10] ADAMIC, L. A. and HUBERMAN, B. A., "Zipf's law and the internet," *Glottometrics*, vol. 3, pp. 143–150, 2002.

[11] ANAGNOSTAKIS, K., SIDIROGLOU, S., AKRITIDIS, P., XINIDIS, K., MARKATOS, E., and KEROMYTIS, A., "Detecting Targeted Attacks Using Shadow Honeypots," in *Proceedings of the 14th Usenix Security Symposium (Security'05)*, (Baltimore, Maryland), August 2005.

[12] BACHER, P., HOLZ, T., KOTTER, M., and WICHERSKI, G., "Know your enemy: Tracking botnets." http://www.honeynet.org/papers/bots/, 2005.

[13] BAECHER, P., KOETTER, M., HOLZ, T., DORNSEIF, M., and FREILING, F., "The nepenthes platform: An efficient approach to collect malware," in *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, (Hamburg), September 2006.

[14] BAILEY, M., OBERHEIDE, J., ANDERSEN, J., MAO, M., JAHANIAN, F., and NAZARIO, J., "Automated classification and analysis of internet malware," in *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'07)*, 2007.

[15] BALASUBRAMANIYAN, J. S., GARCIA-FERNANDEZ, J. O., ISACOFF, D., SPAFFORD, E., and ZAMBONI, D., "An architecture for intrusion detection using autonomous agents," in *Proceedings of the 14th Annual Computer Security Applications Conference (ACSAC'98)*, (Washington, DC, USA), p. 13, IEEE Computer Society, 1998.

[16] BARFORD, P. and YEGNESWARAN, V., "An Inside Look at Botnets." Special Workshop on Malware Detection, Advances in Information Security, Springer Verlag, 2006.

[17] BETHENCOURT, J., FRANKLIN, J., and VERNON, M., "Mapping internet sensors with probe response attacks," in *Proceedings of the 14th USENIX Security Symposium (Security'06)*, 2006.

[18] BINKLEY, J. R. and SINGH, S., "An algorithm for anomaly-based botnet detection," in *Proceedings of USENIX SRUTI'06*, pp. 43–48, July 2006.

[19] BLEEDING EDGE THREATS, "The Bleeding Edge of Snort." `http://www.bleedingsnort.com/`, 2007.

[20] BOHN, K., "Teen questioned in computer hacking probe." CNN, `http://www.cnn.com/2007/TECH/11/29/fbi.botnets/index.html`, 2007.

[21] BREW, C. and MCKELVIE, D., "Word-pair extraction for lexicography," in *Proceedings of NeMLaP '96*, pp. 45–55, 1996.

[22] CHEUNG, S., FONG, M., and LINDQVIST, U., "Modeling multistep cyber attacks for scenario recognition," in *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX III)*, 2003.

[23] CHIANG, K. and LLOYD, L., "A case study of the rustock rootkit and spam bot," in *Proceedings of USENIX HotBots'07*, 2007.

[24] COLLINS, M., SHIMEALL, T., FABER, S., JANIES, J., WEAVER, R., SHON, M. D., and KADANE, J., "Using uncleanliness to predict future botnet addresses,," in *Proceedings of ACM/USENIX Internet Measurement Conference (IMC'07)*, 2007.

[25] COOKE, E., JAHANIAN, F., and MCPHERSON, D., "The zombie roundup: Understanding, detecting, and disrupting botnets," in *Proceedings of USENIX SRUTI'05*, 2005.

[26] CUPPENS, F. and MIEGE, A., "Alert correlation in a cooperative intrusion detection framework," in *Proceedings of IEEE Symposium on Security and Privacy*, 2002.

[27] CYBER-TA, "BotHunter Free Internet Distribution Page." `http://www.cyber-ta.org/BotHunter`, 2008.

[28] CYBER-TA, "SRI Honeynet and BotHunter Malware Analysis." `http://www.cyber-ta.org/Honeynet/`, 2008.

[29] DAGON, D., GU, G., LEE, C., and LEE, W., "A taxonomy of botnet structures," in *Proceedings of the 23 Annual Computer Security Applications Conference (ACSAC'07)*, 2007.

[30] DAGON, D., QIN, X., GU, G., LEE, W., GRIZZARD, J., LEVINE, J., and OWEN, H., "HoneyStat: Local worm detection using honeypots," in *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID'04)*, September 2004.

[31] DAGON, D., ZOU, C., and LEE, W., "Modeling botnet propagation using timezones," in *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, February 2006.

[32] DASWANI, N. and STOPPELMAN, M., "The anatomy of clickbot.a," in *Proceedings of USENIX HotBots'07*, 2007.

[33] DAVIS, J., "Hackers take down the most wired country in europe." WIRED MAGAZINE: ISSUE 15.09, `http://www.wired.com/politics/security/magazine/15-09/ff_estonia`, 2007.

[34] DEGROOT, M. H. and SCHERVISH, M. J., *Probability and Statistics*. Addison-Wesley, 2002.

[35] DINGLEDINE, R., MATHEWSON, N., and SYVERSON, P., "TOR: the second generation onion router," in *Proceedings of the 13th USENIX Security Symposium (Security'04)*, 2004.

[36] ECKMANN, S. T., VIGNA, G., and KEMMERER, R. A., "Statl: An attack language for state-based intrusion detection," *Journal of Computer Security*, vol. 10, 2002.

[37] ELLIS, D. R., AIKEN, J. G., ATTWOOD, K. S., and TENAGLIA, S. D., "A Behavioral Approach to Worm Detection," in *Proceedings of WORM*, 2003.

[38] FOGLA, P., SHARIF, M., PERDISCI, R., KOLESNIKOV, O. M., and LEE, W., "Polymorphic blending attack," in *Proceedings of the 15th USENIX Security Symposium (Security'06)*, 2006.

[39] FORREST, S., HOFMEYR, S., SOMAYAJI, A., and LONGSTAFF, T., "A sense of self for unix processes," in *Proc. IEEE Symposium on Security and Privacy*, pp. 120–128, 1996.

[40] FREILING, F., HOLZ, T., and WICHERSKI, G., "Botnet Tracking: Exploring a Root-cause Methodology to Prevent Denial of Service Attacks," in *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS'05)*, 2005.

[41] GARFINKEL, T., ADAMS, K., WARFIELD, A., and FRANKLIN, J., "Compatibility is Not Transparency: VMM Detection Myths and Realities," in *Proceedings of the 11th Workshop on Hot Topics in Operating Systems (HotOS-XI)*, May 2007.

[42] GIANVECCHIO, S., XIE, M., WU, Z., and WANG, H., "Measurement and classification of humans and bots in internet chat," in *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[43] GOEBEL, J. and HOLZ, T., "Rishi: Identify bot contaminated hosts by irc nickname evaluation," in *Proceedings of USENIX HotBots'07*, 2007.

[44] GRIZZARD, J. B., SHARMA, V., NUNNERY, C., KANG, B. B., and DAGON, D., "Peer-to-peer botnets: Overview and case study," in *Proceedings of USENIX HotBots'07*, 2007.

[45] GU, G., PERDISCI, R., ZHANG, J., and LEE, W., "BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection," in *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[46] GU, G., PORRAS, P., YEGNESWARAN, V., FONG, M., and LEE, W., "BotHunter: Detecting malware infection through ids-driven dialog correlation," in *Proceedings of the 16th USENIX Security Symposium (Security'07)*, 2007.

[47] GU, G., SHARIF, M., QIN, X., DAGON, D., LEE, W., and RILEY, G., "Worm detection, early warning and response based on local victim information," in *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, (Washington, DC, USA), pp. 136–145, IEEE Computer Society, 2004.

[48] GU, G., ZHANG, J., and LEE, W., "BotSniffer: Detecting botnet command and control channels in network traffic," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, 2008.

[49] HALKIDI, M., BATISTAKIS, Y., and VAZIRGIANNIS, M., "On clustering validation techniques," *J. Intell. Inf. Syst.*, vol. 17, no. 2-3, pp. 107–145, 2001.

[50] HOLZ, T. and RAYNAL, F., "Detecting honeypots and other suspicious environments," in *Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop (IAW'05)*, 2005.

[51] HOLZ, T., GORECKI, C., RIECK, K., and FREILING, F. C., "Detection and mitigation of fast-flux service networks," in *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, 2008.

[52] HOLZ, T., STEINER, M., DAHL, F., BIERSACK, E., and FREILING, F., "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'08)*, 2008.

[53] IANELLI, N. and HACKWORTH, A., "Botnets as a vehicle for online crime." http://www.cert.org/archive/pdf/Botnets.pdf, 2005.

[54] IGLUN, K., KEMMERER, R. A., and PORRAS, P. A., "State transition analysis: A rule-based intrusion detection system," *IEEE Transactions on Software Engineering*, vol. 21, 1995.

[55] JAIN, A. K., MURTY, M. N., and FLYNN, P. J., "Data clustering: a review," *ACM Computer Survey*, vol. 31, no. 3, pp. 264–323, 1999.

[56] JIANG, X. and XU, D., "Profiling Self-Propagating Worms Via Behavioral Footprinting," in *Proceedings of ACM WORM'06* , 2006.

[57] JUNG, J., PAXSON, V., BERGER, A. W., and BALAKRISHNAN, H., "Fast Portscan Detection Using Sequential Hypothesis Testing," in *IEEE Symposium on Security and Privacy 2004*, (Oakland, CA), May 2004.

[58] JUNG, J., SCHECHTER, S. E., and BERGER, A. W., "Fast detection of scanning worm infections," in *Proceedings of RAID'2004*, September 2004.

[59] KARASARIDIS, A., REXROAD, B., and HOEFLIN, D., "Wide-scale botnet detection and characterization," in *Proceedings of USENIX HotBots'07*, 2007.

[60] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., and KAASHOEK, M. F., "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 3, pp. 263–297, 2000.

[61] KREBS, B., "Storm worm dwarfs world's top supercomputers." `http://blog.washingtonpost.com/securityfix/2007/08/storm_worm_dwarfs_worlds_top_s_1.html`, 2007.

[62] KREBS, B., "Kraken spawns a clash of the titans." Washington Post,`http://blog.washingtonpost.com/securityfix/2008/04/kraken_creates_a_clash_of_the.html`, 2008.

[63] LEE, W., WANG, C., and DAGON, D., *Botnet Detection: Countering the Largest Security Threat (Advances in Information Security)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007.

[64] LEMOS, R., "Bot software looks to improve peerage." Http://www.securityfocus.com/news/11390, 2006.

[65] LEYDEN, J., "Zombie pcs spew out 80% of spam." The Register, http://www.theregister.co.uk/2004/06/04/trojan_spam_study/, 2004.

[66] LIVADAS, C., WALSH, R., LAPSLEY, D., and STRAYER, W. T., "Using machine learning techniques to identify botnet traffic," in *Proceedings of the 2nd IEEE LCN Workshop on Network Security (WoNS'2006)*, 2006.

[67] MALAN, D. J., *Rapid detection of botnets through collaborative networks of peers*. PhD thesis, Harvard University, Cambridge, MA, USA, 2007.

[68] MAYMOUNKOV, P. and MAZIERES, D., "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

[69] MITRE CORPORATION, "CVE-2006-3439." http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-3439.

[70] MOORE, D., "Network telescopes." http://www.caida.org/outreach/presentations/2002/usenix_sec/, 2002.

[71] MOORE, D., VOELKER, G., and SAVAGE, S., "Inferring Internet Denial-of-Service Activity," in *Proceedings of the 10th USENIX Security Symposium*, 2001.

[72] MYERS, L., "Aim for bot coordination," in *Virus Bulletin Conference*, 2006.

[73] NING, P., CUI, Y., and REEVES, D. S., "Constructing attack scenarios through correlation of intrusion alerts," in *Proceedings of the 9th ACM Conference on Computer & Communications Security (CCS'02)*, 2002.

[74] PAXSON, V., "Bro: A System for Detecting Network Intruders in Real Time," in *Proceedings of the 7th USENIX Security Symposium*, 1998.

[75] PELLEG, D. and MOORE, A. W., "X-means: Extending k-means with efficient estimation of the number of clusters," in *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*, (San Francisco, CA, USA), pp. 727–734, Morgan Kaufmann Publishers Inc., 2000.

[76] PERDISCI, R., GU, G., and LEE, W., "Using an ensemble of one-class svm classifiers to harden payload-based anomaly detection systems," in *Proceedings of the IEEE International Conference on Data Mining (ICDM'06)*, December 2006.

[77] PORRAS, P., SAIDI, H., and YEGNESWARAN, V., "A multi-perspective analysis of the storm (peacomm) worm," tech. rep., Computer Science Laboratory, SRI International, October 2007.

[78] PORRAS, P. A. and NEUMANN, P. G., "EMERALD: Event monitoring enabling responses to anomalous live disturbances," in *Proc. 20th NIST-NCSC National Information Systems Security Conference*, pp. 353–365, 1997.

[79] PORRAS, P., "Privacy-enabled Global Threat Monitoring," *IEEE Security and Privacy Magazine*, vol. 4, pp. 60–63, Nov/Dec 2006.

[80] PROVOS, N., "A virtual honeypot framework," in *Proceedings of 13th USENIX Security Symposium (Security'04)*, August 2004.

[81] PROVOS, N., MAVROMMATIS, P., RAJAB, M., and MONROSE, F., "All your iframes point to us," in *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[82] RAJAB, M., ZARFOSS, J., MONROSE, F., and TERZIS, A., "A multi-faceted approach to understanding the botnet phenomenon," in *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference (IMC'06)*, (Brazil), October 2006.

[83] RAMACHANDRAN, A., FEAMSTER, N., and VEMPALA, S., "Filtering spam with behavioral blacklisting," in *Proceedings of ACM Conference on Computer and Communications Security (CCS'07)*, 2007.

[84] RAMACHANDRAN, A. and FEAMSTER, N., "Understanding the network-level behavior of spammers," in *Proceedings of ACM SIGCOMM'06*, 2006.

[85] RAMACHANDRAN, A., FEAMSTER, N., and DAGON, D., "Revealing botnet membership using DNSBL counter-intelligence," in *Proceedings of USENIX SRUTI'06*, 2006.

[86] ROESCH, M., "Snort - lightweight intrusion detection for networks," in *Proceedings of USENIX LISA'99*, 1999.

[87] ROYAL, P., HALPIN, M., DAGON, D., EDMONDS, R., and LEE, W., "Polyunpack: Automating the hidden-code extraction of unpack-executing malware," in *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06)*, (Washington, DC, USA), pp. 289–300, IEEE Computer Society, 2006.

[88] SCHULTZ, M. G., ESKIN, E., ZADOK, E., and STOLFO, S. J., "Data mining methods for detection of new malicious executables," in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001.

[89] SMALL, S., MASON, J., MONROSE, F., PROVOS, N., and STUBBLEFIELD, A., "To catch a predator: A natural language approach for eliciting malicious payloads," in *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[90] SNAPP, S. R., BRENTANO, J., DIAS, G. V., GOAN, T. L., HEBERLEIN, L. T., LIN HO, C., LEVITT, K. N., MUKHERJEE, B., SMAHA, S. E., GRANCE, T., TEAL,

D. M., and MANSUR, D., "DIDS (distributed intrusion detection system) - motivation, architecture, and an early prototype," in *Proceedings of the 14th National Computer Security Conference*, (Washington, DC), pp. 167–176, 1991.

[91] SOMMER, R. and PAXSON, V., "Enhancing byte-level network intrusion detection signatures with context," in *Proceedings of ACM Conference on Computer and Communication Security (CCS'03)*, (Washington, DC), October 2003.

[92] SOPHOS INC., "W32/IRCBot-TO." `http://www.sophos.com/virusinfo/analyses/w32ircbotto.html`, 2007.

[93] STANIFORD, S., PARXON, V., and N.WEAVER, "How to Own the Internet in Your Spare Time," in *Proceedings of 2002 USENIX Security Symposimum*, 2002.

[94] STANIFORD, S., HOAGLAND, J., and MCALERNEY, J., "Practical Automated Detection of Stealthy Portscans," in *Journal of Computer Security*, 2002.

[95] STANIFORD-CHEN, S., CHEUNG, S., CRAWFORD, R., DILGER, M., FRANK, J., HOAGLAND, J., LEVITT, K., WEE, C., YIP, R., and ZERKLE, D., "Grids–a graph based intrusion detection system for large networks," in *19th National Information Systems Security Conference*, 1996.

[96] STEWART, J., "Bobax trojan analysis." `http://www.secureworks.com/research/threats/bobax/`, 2004.

[97] STINSON, E. and MITCHELL, J. C., "Characterizing bots' remote control behavior," in *Proceedings of the 4th GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'07)*, 2007.

[98] STRAYER, W. T., WALSH, R., LIVADAS, C., and LAPSLEY, D., "Detecting botnets with tight command and control," in *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN'06)*, 2006.

[99] SZOR, P., *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, 2005.

[100] TEMPLETON, S. and LEVITT, K., "A requires/provides model for computer attacks," in *New Security Paradigms Workshop*, 2000.

[101] TURING, A., "Computing machinery and intelligence," in *Mind, Vol.59, pp 433-460*, 1950.

[102] VALDES, A. and SKINNER, K., "Probabilistic alert correlation," in *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'01)*, pp. 54–68, 2001.

[103] VALEUR, F., VIGNA, G., C.KRUEGEL, and KEMMERER, R., "A Comprehensive Approach to Intrusion Detection Alert Correlation," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, pp. 146–169, July-September 2004.

[104] VIGNA, G. and KEMMERER, R., "NetSTAT: A Network-based Intrusion Detection Approach," in *Proceedings of the* 14^{th} *Annual Computer Security Applications Conference (ACSAC'98)*, 1998.

[105] VIJAYAN, J., "Teen used botnets to push adware to hundreds of thousands of pcs." `http://www.computerworld.com/action/article.do?command=viewArticleBasic&articleId=9062839`, 2008.

[106] VOGT, R., AYCOCK, J., and JACOBSON, M., "Army of botnets," in *Proceedings of the 14th Network and Distributed System Security Symposium (NDSS'07)*, 2007.

[107] VON AHN, L., BLUM, M., HOPPER, N., and LANGFORD, J., "CAPTCHA: Using hard AI problems for security," in *Proceedings of Eurocrypt*, pp. 294–311, 2003.

[108] WALD, A., *Sequential Analysis*. Dover Publications, 2004.

[109] WANG, K., PAREKH, J. J., and STOLFO, S., "Anagram: A content anomaly detector resistant to mimicry attack," in *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, 2006.

[110] WANG, K. and STOLFO, S., "Anomalous payload-based network intrusion detection," in *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'04)*, 2004.

[111] WANG, K. and STOLFO, S., "Anomalous payload-based worm detection and signature generation," in *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'05)*, 2005.

[112] WANG, P., SPARKS, S., and ZOU, C. C., "An advanced hybrid peer-to-peer botnet," in *Proceedings of USENIX HotBots'07*, 2007.

[113] WANG, Y.-M., BECK, D., JIANG, X., ROUSSEV, R., VERBOWSKI, C., CHEN, S., and KING, S., "Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities," in *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, February 2006.

[114] WARD, M., "More than 95% of e-mail is 'junk'." `http://news.bbc.co.uk/1/hi/technology/5219554.stm`, 2006.

[115] WEAVER, N., STANIFORD, S., and PAXSON, V., "Very fast containment of scanning worms," in *Proceedings of 13 USENIX Security Symposium (Security'04)*, October 2004.

[116] WEBER, T., "Criminals 'may overwhelm the web'." `http://news.bbc.co.uk/1/hi/business/6298641.stm`, 2007.

[117] WEHNER, S., "Analyzing worms and network traffic using compression," *Journal of Computer Security*, vol. 15, no. 3, pp. 303–320, 2007.

[118] WERNER, T., "PE Hunter." `http://honeytrap.mwcollect.org/pehunter`, 2007.

[119] WHYTE, D., VAN OORSCHOT, P., and KRANAKIS, E., "Exposure maps: Removing reliance on attribution during scan detection," in *Proceedings of 1st USENIX Workshop on Hot Topics in Security (HotSec'06)*, 2006.

[120] WU, J., VANGALA, S., GAO, L., and KWIAT, K., "An efficient architecture and algorithm for detecting worms with various scan techniques," in *Proceedings of NDSS'2004*, February 2004.

[121] XIE, M., YIN, H., and WANG, H., "An effective defense against email spam laundering," in *Proceedings of ACM Conference on Computer and Communication Security (CCS'06)*, 2006.

[122] XIE, Y., KIM, H.-A., O'HALLARON, D., REITER, M. K., and ZHANG, H., "Seurat: A pointillist approach to anomaly detection," in *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'04)*, 2004.

[123] YANG, J., NING, P., WANG, X. S., and JAJODIA, S., "CARDS: A distributed system for detecting coordinated attacks," in *Proceedings of IFIP TC11 Sixteenth Annual Working Conference on Information Security (SEC)*, 2000.

[124] YEN, T.-F. and REITER, M. K., "Traffic aggregation for malware detection," in *Proceedings of the Fifth GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'08)*, 2008.

[125] ZHUANG, L., DUNAGAN, J., SIMON, D. R., WANG, H. J., OSIPKOV, I., HULTEN, G., and TYGAR, J., "Characterizing botnets from email spam records," in *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*, 2008.

[126] ZHUGE, J., HOLZ, T., HAN, X., GUO, J., and ZOU, W., "Characterizing the irc-based botnet phenomenon." Peking University & University of Mannheim Technical Report, 2007.

[127] ZOU, C. C., GAO, L., GONG, W., and TOWSLEY, D., "Monitoring and early warning for internet worms," in *Proceedings of ACM CCS'2003*, October 2003.

[128] ZOU, C. C. and CUNNINGHAM, R., "Honeypot-aware advanced botnet construction and maintenance," in *International Conference on Dependable Systems and Networks (DSN'06)*, 2006.

VITA

Guofei Gu was born in Changshu, JiangSu Province, China. In 2000, he received his bachelor's degree in computer science from Nanjing University of Posts and Telecommunications, Nanjing, China. In 2003, he was awarded master's degree in computer science from Fudan University, Shanghai, China. Subsequently, he joined the Computer Science Ph.D program in the College of Computing at the Georgia Institute of Technology in Fall 2003. During his Ph.D. study, he conducted research under the advisement of Dr. Wenke Lee in the area of network and system security, in particular, in intrusion detection, malware detection, defense, and analysis. He was also collaborated with SRI International during summer internships.