

2010

Towards secure message systems

Mengjun Xie

College of William & Mary - Arts & Sciences

Follow this and additional works at: <https://scholarworks.wm.edu/etd>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Xie, Mengjun, "Towards secure message systems" (2010). *Dissertations, Theses, and Masters Projects*. Paper 1539623341.

<https://dx.doi.org/doi:10.21220/s2-91s6-5e87>

This Dissertation is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Dissertations, Theses, and Masters Projects by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

Towards Secure Message Systems

Mengjun Xie

Lu'an, Anhui Province, China

Master of Engineering, East China Normal University, 2002

Bachelor of Engineering, East China Normal University, 1999

**A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy**

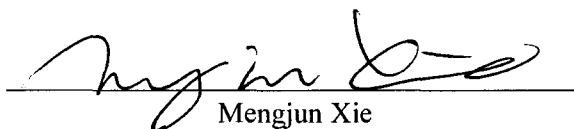
Department of Computer Science

**The College of William and Mary
January 2010**

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy



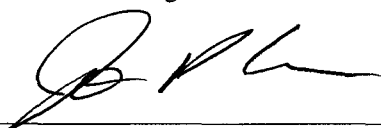
Mengjun Xie

Approved by the Committee, November 2009



Committee Chair

Associate Professor Haining Wang, Computer Science
The College of William and Mary



Associate Professor Phil Kearns, Computer Science
The College of William and Mary



Associate Professor Qun Li, Computer Science
The College of William and Mary



Associate Professor Weizhen Mao, Computer Science
The College of William and Mary



Professor Chi-Kwong Li, Mathematics
The College of William and Mary

ABSTRACT PAGE

Message systems, which transfer information from sender to recipient via communication networks, are indispensable to our modern society. The enormous user base of message systems and their critical role in information delivery make it the top priority to secure message systems. This dissertation focuses on securing the two most representative and dominant messages systems—e-mail and instant messaging (IM)—from two complementary aspects: defending against unwanted messages and ensuring reliable delivery of wanted messages.

To curtail unwanted messages and protect e-mail and instant messaging users, this dissertation proposes two mechanisms DBSpam and HoneyIM, which can effectively thwart e-mail spam laundering and foil malicious instant message spreading, respectively. DBSpam exploits the distinct characteristics of connection correlation and packet symmetry embedded in the behavior of spam laundering and utilizes a simple statistical method, Sequential Probability Ratio Test, to detect and break spam laundering activities inside a customer network in a timely manner. The experimental results demonstrate that DBSpam is effective in quickly and accurately capturing and suppressing e-mail spam laundering activities and is capable of coping with high speed network traffic. HoneyIM leverages the inherent characteristic of spreading of IM malware and applies the honeypot technology to the detection of malicious instant messages. More specifically, HoneyIM uses decoy accounts in normal users' contact lists as honeypots to capture malicious messages sent by IM malware and suppresses the spread of malicious instant messages by performing network-wide blocking. The efficacy of HoneyIM has been validated through both simulations and real experiments.

To improve e-mail reliability, that is, prevent losses of wanted e-mail, this dissertation proposes a collaboration-based autonomous e-mail reputation system called CARE. CARE introduces inter-domain collaboration without central authority or third party and enables each e-mail service provider to independently build its reputation database, including frequently contacted and unacquainted sending domains, based on the local e-mail history and the information exchanged with other collaborating domains. The effectiveness of CARE on improving e-mail reliability has been validated through a number of experiments, including a comparison of two large e-mail log traces from two universities, a real experiment of DNS snooping on more than 36,000 domains, and extensive simulation experiments in a large-scale environment.

Table of Contents

Acknowledgments	ix
List of Tables	x
List of Figures	xi
1 Introduction	2
1.1 Contributions	7
1.1.1 Thwarting E-mail Spam Laundering	7
1.1.2 Countering Malicious Instant Messages	9
1.1.3 Improving E-mail Reliability	10
1.2 Organization	11
2 Background	12
2.1 E-mail Spamming Mechanisms	12
2.1.1 Spam Laundering Mechanisms	12
2.1.2 Other Spamming Approaches	14
2.2 Instant Messaging Malware	15
3 Thwarting E-mail Spam Laundering	19

3.1	Proxy-based Spam Behavior	20
3.1.1	Laundry Path of Proxy-based Spamming	21
3.1.2	Connection Correlation	23
3.1.3	Packet Symmetry	23
3.2	Working Mechanism of DBSpam	27
3.2.1	Deployment of DBSpam	27
3.2.2	Design Choices and Overview	28
3.2.3	Sequential Probability Ratio Testing	30
3.2.4	SPRT Detection Algorithm	33
3.2.5	Noise Reduction	37
3.3	System Evaluation	40
3.3.1	Data Collection	40
3.3.2	Detection Time	42
3.3.3	Detection Accuracy	44
3.3.4	Resource Consumption	47
3.3.5	Suppressing Spam Activities	49
3.4	Potential Evasions	52
3.5	Related Work	54
3.5.1	Recipient-based Techniques	54
3.5.2	Sender-based Techniques	57
3.5.3	HoneySpam	58
3.6	Summary	59
4	Countering Malicious Instant Messages	60

4.1	Related Work	61
4.2	HoneyIM Framework	62
4.2.1	Overview	63
4.2.2	Design Issues	65
4.2.3	System Components	66
4.2.4	Deployment	71
4.2.5	Prototype	72
4.3	Evaluation	73
4.3.1	Simulation	73
4.3.2	Real Experiment	78
4.4	Discussion	79
4.5	Summary	82
5	Improving E-mail Reliability	83
5.1	Related Work	84
5.2	Motivation	86
5.3	System Design	90
5.3.1	Rating Issues	92
5.3.2	Local E-mail History	92
5.3.3	Reputation Database	93
5.3.4	Simple E-mail Reputation Protocol (SERP)	98
5.4	System Evaluation	101
5.4.1	Log-based Experiment	101
5.4.2	DNS-based Experiment	102

5.4.3	Simulation Experiments	104
5.5	Summary	110
6	Conclusions and Future Work	111
6.1	Future Work	112
	Bibliography	115
	Vita	123

*To my parents, for their unwavering love and support
throughout this and all my adventures.*

ACKNOWLEDGMENTS

First and foremost, I thank my advisor, Dr. Haining Wang, for his guidance, advice, support, and encouragement through my PhD study. His focus on significant research problems and passion for high quality research have deeply shaped my research attitude and greatly helped me to build a solid foundation for my future career.

I also thank Dr. Phil Kearns, Dr. Chi-Kwong Li, Dr. Qun Li, and Dr. Weizhen Mao for serving on my thesis committee and providing me with invaluable comments and advice through the whole process of thesis writing.

I am grateful to the staff of the Department of Computer Science for their assistance. Special thanks go to Vanessa Godwin. She has made my life in William and Mary much easier.

I am indebted to the past and current PhD students in Computer Science Department, from whom I got a lot of help, on both study and life. I have enjoyed collaborating with Zhenyu Wu, Chuan Yue, and Steven Gianvecchio. I feel fortunate to have them as my friends and teammates.

Finally, I thank my parents, Daoming Xie and Kaiyun Jiang. They are the source of strengths for me to overcome the difficulties on the way of pursuing my PhD. This thesis is dedicated to them.

List of Tables

3.1	Trace information	41
3.2	Distribution of $N H_1$	43
3.3	False positives and false negatives of SPRT	46
3.4	Overall false positives of DBSpam ($\Delta = 2s$)	48
3.5	Resource consumption	49
3.6	False positive comparisons ($M = 5, K = 4, \Delta = 2s$)	53
5.1	Summary of notations	95

List of Figures

3.1	Scenario of proxy-based spamming	21
3.2	Time-line of spamming processes for single proxy (left) and proxy chain (right) .	24
3.3	Example of reply round and TCP correlation	25
3.4	$E[N H_1]$ vs. θ_0 and α^* ($\theta_1 = 0.99, \beta^* = 0.01$)	37
3.5	$\Pr(X \geq K)$ vs. p and (M, K)	38
3.6	Distribution of $N H_0$	44
3.7	CDF of detection time for SPRT	45
3.8	Comparison of number of messages sent out before and after throttling	50
3.9	Comparison of TCP packet numbers before and after blocking	51
4.1	Working mechanism of HoneyIM	64
4.2	Framework of HoneyIM	66
4.3	Relations between HoneyIM coverages and infected user percentages	74
4.4	Comparisons among mean curves	77
4.5	Effect comparisons between HoneyIM and IM throttling	78
4.6	Effects of randomly selecting infection targets on HoneyIM	81
5.1	Number of newly-appeared senders per day	87
5.2	CDF of good-ratio for spam and nonspam (good) e-mail	89

5.3	Architecture of CARE system	91
5.4	Procedure of a successful mutual agreement establishment via SERP	99
5.5	Percentage of newly-appeared IP addresses that have been recorded by the other university over all newly-appeared IP addresses in daily logs	101
5.6	Number of DNS cache hits for 25 .edu domains	103
5.7	Percentage of newly-appeared nonspam domains that are covered after using CARE in each day	106
5.8	Percentage of newly-appeared spam domains that are covered after using CARE in each day	107
5.9	Percentage of more nonspam messages being directly accepted after using CARE in each day	108
5.10	Percentage of more spam messages being directly rejected after using CARE in each day	109

Towards Secure Message Systems

Chapter 1

Introduction

Message systems, which transfer information from sender to recipient via communication networks, are indispensable to our modern society. E-mail, the most representative message system, is regarded as one of the core and most successful Internet applications. According to the survey done in April 2009 by Pew Internet and American Life Project [12], 90% of U.S. Internet users have the experience of sending or reading e-mail. An e-mail statistics report published in May 2009 [11] estimates that the number of worldwide e-mail users in 2009 is over 1.4 billion and that around 247 billion messages are delivered worldwide every day in 2009. Message systems are used in every sector of our society (e.g., business, education, government) and messages are accessed from anywhere (e.g., PC, laptop, cell phone, PDA, and TV) at anytime.

In general, there are two types of message systems in terms of message transfer synchrony: online message systems and offline message systems. Online message systems usually require message sender and receiver to stay online, that is, keep synchronous, during a message transfer, while offline message systems do not. E-mail is a typical example of the former and instant messaging (IM) is a representative system of the latter. Because e-mail and IM perfectly complement

each other in usage, interoperation and intercommunication between e-mail and IM systems have been attained, which blurs the boundary between e-mail and IM from the perspective of end users. For example, most of major e-mail service providers (ESPs) such as Yahoo! mail, Hotmail, and Gmail have already integrated the instant messaging functionality into their Web-based e-mail services and have added e-mail functionality in their IM client programs. Due to their representativeness and dominance among message systems in practice, we focus on e-mail and instant messaging in this dissertation.

The enormous user base of message systems and their critical role in information delivery make it the top priority to secure message systems. Since the commercialization of Internet, network environments are no longer friendly. Networked applications are constantly subjected to various types of attacks, and there is no exception to Internet-based message systems. The prevailing type of attacks against both e-mail and IM is unwanted messages, which range from bogus commercial advertisements, crafted scam and phishing messages, to virus. In this dissertation, we term unwanted messages in general as spam. Recent years have seen steady growth of e-mail spam volume [1, 76] and frequent outbreaks of malware spread via e-mail and IM [9, 10]. Unwanted messages severely endanger the usability and security of e-mail and IM.

Thanks to spam filters, flood of unwanted messages is effectively dammed before the front door of the inboxes of end users. However, another significant problem—loss of e-mail—has emerged along with spam filters. Aggressive spam filters may cause loss of e-mail. A recent study on e-mail loss [13] reveals that the e-mail accounts with spam filtering lost significantly more legitimate messages than the e-mail accounts without spam filtering¹. Many anecdotal reports including the loss of e-mail submissions discussed in the “end2end” mail-list [106] also indicate

¹The messages are neither in the inbox nor in the spam folder.

the existence of e-mail loss due to spam filtering. In addition, as system overloading can cause loss of e-mail [7], being computationally intensive, spam filtering could overload e-mail systems when an influx of e-mail messages occurs and therefore result in loss of legitimate messages.

Therefore, securing message systems requires (1) defending against unwanted messages and (2) ensuring reliable delivery of wanted messages. These two tasks are the two sides of one coin; together, they comprise the main theme of this dissertation: *secure message systems*. Next, we consider the challenges in each of these two tasks and highlight the basic ideas of our corresponding solutions.

1) Defending against unwanted messages

Content-based spam filters and IP address-based blacklists are the two most popular mechanisms that protect e-mail users from unwanted e-mail messages in practice. Content-based spam filters distinguish e-mail spam from legitimate e-mail by exploiting distinctive content features of spam messages [35, 90]. A number of classification techniques ranging from statistics to machine learning have been applied to capture content features of e-mail spam [19, 112]. IP address-based blacklists record IP addresses of the identified hosts that sent spam. In practice, these blacklists are often called DNS blackhole lists (DNSBLs) as they are mostly distributed through DNS. Spam filtering is applied after receiving a new message while IP address-based blacklisting is applied before message content is received.

Although spam filters are widely deployed and effective in keeping e-mail spam out of inbox, they lack the ability of suppressing spam in the first place. Therefore, spam still wastes a lot of resources on the Internet and at receiving servers even if they are filtered away from inboxes of end users. DNSBLs can block spamming promptly. However, they suffer from insufficient coverage and responsiveness [27, 71–73]. As spammers frequently change spamming hosts and manage to

keep spamming activities stealthy, many spamming hosts are missed by DNSBLs. To stem the tide of spam that keeps growing [1], new anti-spam mechanisms that can quickly and accurately identify and suppress spamming activities are highly demanded.

Different from e-mail systems, instant messaging systems have built-in authentication and authorization mechanisms, effectively preventing spamming from unknown sources. Unfortunately, spamming approaches have also evolved. Social engineering tricks have been largely employed in sending unwanted messages in IM systems. Even worse, IM spamming (usually called “spim”) is mostly done by IM malware, which often results in breach of system security and causes much more damage to IM users. Detection of malicious instant messages is hard due to the legitimacy of message sources and camouflage of message contents. Previously proposed mechanisms for countering malicious instant messages mainly focus on delaying the spread of malicious messages by throttling message sending on all IM users [54, 100, 108]. However, it is ideal to quickly and accurately detect and stop the spread of IM malware at the beginning.

Therefore, mechanisms that can foil spamming activities are in urgent need to curtail unwanted messages and protect e-mail and IM users. This dissertation presents two mechanisms—DBSpam and HoneyIM—to effectively thwart e-mail spam laundering and malicious instant message spreading, respectively. Since spamming activities are abnormal compared to legitimate message sending, our general approach is distilling the distinct behaviors that are embedded in spamming activities and exploiting them to detect occurrences of spamming in real-time. For example, DBSpam exploits the characteristic of packet symmetry in e-mail spam laundering and HoneyIM leverages the characteristic of message sending in IM malware propagating. Both DBSpam and HoneyIM can achieve high detection accuracy and short detection time. More importantly, they do not require any change to the protocols used in e-mail and IM and are fully compatible with

existing protection techniques.

2) Ensuring delivery of wanted messages

Whitelisting legitimate e-mail addresses to bypass spam filtering is a simple yet widely-adopted approach to preventing loss of wanted messages. E-mail address whitelists usually work with e-mail authentication mechanisms such as SPF [109] and DKIM [14] to prevent attacks of forged e-mail addresses. A significant problem of e-mail address whitelists is that they cannot provide protection for legitimate messages originated from “newly-appeared” addresses, that is, the addresses not in the whitelists. Although approaches such as [21,24,30] that propagate address whitelists among friends can alleviate the coverage problem, whitelisting at e-mail address level is not scalable in general.

A domain-level e-mail reputation system is a more general and scalable solution to preventing e-mail loss. E-mail from reputable domains can be directly accepted. However, as indicated in [98, 101] and confirmed by our measurement study (see Section 5.2), newly-appeared sending domains are common and significant to average e-mail service providers; thus local information is insufficient and collaboration is needed for building a high-quality reputation system. One way to achieve collaboration is using a centralized server to collect information and derive reputation, as proposed by Singaraju and Kang [81]. However, this approach involves the complication of trust management.

In contrast, this dissertation explores a different dimension in the design space. To ease system deployment and maintenance, we propose an autonomous reputation system named CARE that involves no central server or third party. Within the framework of CARE, each domain independently builds its reputation database based on the local e-mail history and information exchanged with other collaborating domains; each domain also has full control of choosing their collaboration

domains. By doing so, CARE achieves both effectiveness and flexibility.

1.1 Contributions

The contributions of this dissertation are summarized as follows.

1.1.1 Thwarting E-mail Spam Laundering

- We have thoroughly studied the mechanisms of e-mail spam laundering, an important spamming method in which spam proxies are employed to disguise the identities of spam origins. We have distilled the unique characteristics of connection correlation and packet symmetry from the behavior of spam laundering by analyzing the protocol semantics of SMTP and timing causality in spam laundering. To our best knowledge, our study is the first to reveal the distinct characteristic of e-mail spam laundering, that is, packet symmetry.
- Based on the packet symmetry exhibited in spam laundering, we have developed a simple yet effective technique, DBSpam, to detect and break spam laundering activities inside a customer network in a timely manner. DBSpam is designed to be deployed at a network vantage point such as an edge router or gateway that connects the network to the Internet. Monitoring the bidirectional traffic passing through a network gateway, DBSpam exploits the packet symmetry characteristic and utilizes a simple statistical method, Sequential Probability Ratio Test, to capture the TCP connections involved in spam laundering, single out the spam proxies, and uncover the spam sources behind them in a timely manner. To balance the goals of promptness and accuracy, we introduce a noise-reduction technique into DBSpam, after which the laundering path can be identified more accurately. DBSpam pro-

vides two spam suppressing methods: rate-limit throttling and blocking, and activates the user-selected suppressing method immediately after a laundering activity is detected.

- DBSpam pushes the defense line towards spam source. DBSpam is the first system that foils e-mail spam laundering without the cooperation at e-mail receiving side. Therefore, DBSpam greatly benefits not only e-mail users but also victim Internet Service Providers (ISPs). DBSpam enables an ISP to accurately detect spam laundering activities and spam proxies inside its customer networks. The quick responsiveness of DBSpam offers the ISP an opportunity to suppress laundering activities and quarantine identified spam proxies in real-time. Being a stand-alone system, DBSpam is incrementally deployable over the Internet.
- Distinctive from content-based spam filtering techniques, DBSpam is lightweight in that its detection technique does not need to scan message contents. Moreover, DBSpam has very few assumptions about the connections between a spammer and its proxies. DBSpam works even if (1) these connections are encrypted and the message contents are compressed; and (2) a spammer uses proxy chains inside the monitored network.
- DBSpam complements existing anti-spam techniques. Furthermore, DBSpam can facilitate spam filtering. This is because once spam laundering is detected, fingerprinting spam messages at the sender side is viable and spam signatures may be distributed to spam filters elsewhere.
- We have implemented a prototype of DBSpam using *libpcap* on Linux and extensively evaluated its effectiveness and performance on both detecting and suppressing spam laundering through trace-based experiments. We collected a set of traces from a real network

consisting of over 7,000 users and the traces are significant (30+ gigabytes and 7+ hours). Our experiments show that (1) DBSpam achieves user-expected accuracy; (2) Detections of spam laundering are no more than ten seconds and 95% of them are within five seconds; (3) DBSpam is lightweight in terms of CPU and memory consumption and therefore is capable of working at high-speed networks.

1.1.2 Countering Malicious Instant Messages

- We have developed a generic framework, HoneyIM, to automatically detect and suppress the spread of malicious instant messages in an enterprise-like network. HoneyIM is the first system that exploits the inherent characteristic of IM malware spreading and applies the honeypot technology to the detection of malicious instant messages. HoneyIM uses decoy accounts in normal users' contact lists as sensors (i.e., honeypots) to capture malicious messages sent by IM malware. By doing so, HoneyIM can achieve almost zero false positive. With accurate detection, HoneyIM suppresses the spread of malicious instant messages by performing network-wide blocking. HoneyIM can also notify network administrators of attack information in real-time for system quarantine and recovery. The core design of HoneyIM is generic and can be applied to a network that uses either private (enterprise) or public IM services, which is difficult to achieve for previously proposed IM protection approaches.
- We have implemented a prototype of HoneyIM for public IM services, based on open-source IM client Pidgin [8] and client honeypot Capture [93]. We have validated the efficacy of HoneyIM through both simulations and real experiments. The simulations show that even only a small portion, e.g., 5%, of IM users in the network have decoys in their contact lists, HoneyIM can detect the spread of malicious instant messages as early as after 0.4% of

IM users are infected on average. The experimental results demonstrate that the prototype system succeeds in detection, suppression, and notification of IM malware within seconds.

1.1.3 Improving E-mail Reliability

- We have conducted a measurement study on the dynamics of e-mail sending servers and sending domains to investigate whether local e-mail history information is sufficient for average e-mail service providers to build a reputation system with good performance. After studying 151-day e-mail logs collected from our campus e-mail servers, we find that the number of newly-appeared sending parties, in terms of both sending servers and sending domains, is significant. Therefore, our study indicates that only local information may not suffice for building a high-quality reputation system. Meanwhile, our study also confirms that rating e-mail sending parties by their long-term behaviors is feasible and beneficial.
- We have designed a collaboration-based autonomous e-mail reputation system called CARE that aims to significantly improve e-mail reliability. CARE works at domain level and rates both spam domains and nonspam domains. Within the framework of CARE, each e-mail service provider independently builds its reputation database, including both frequently contacted and unacquainted sending domains, based on the local e-mail history and the information exchanged with other collaborating domains. CARE examines the trustworthiness of the e-mail histories obtained from collaborators by correlating them with the local history, and integrates both local and remote information to derive the reputation of remote domains. As there is no hierarchical dependence in system architecture and no requirement of third party, CARE is friendly to incremental deployment.
- We have conducted a number of experiments to validate the effectiveness of CARE on im-

proving e-mail reliability. By comparing two large e-mail log traces from two universities and conducting a real experiment of DNS snooping on more than 36,000 domains, we show that the use of collaboration among different domains in CARE can largely increase the coverage of reputation system. By performing extensive simulation experiments in a large-scale environment, we further demonstrate that CARE is effective in improving the reliability and quality of e-mail service by accepting more nonspam messages and rejecting more spam messages.

1.2 Organization

This dissertation is organized as follows: We present the background information of e-mail and instant messaging spamming in Chapter 2. In Chapter 3, we first delineate the behavior of e-mail spam laundering and then detail the working mechanism of DBSpam followed by the evaluation of DBSpam. We also discuss the robustness of DBSpam against potential evasions and survey related anti-spam techniques in Chapter 3. We present the design, implementation, and evaluation of HoneyIM in Chapter 4. In Chapter 5, we first present a measurement study that motivates the CARE system and then detail the design and evaluation of the CARE system. Finally, we conclude this dissertation and outline our future work in Chapter 6.

Chapter 2

Background

In this chapter, we present the background information about e-mail spamming mechanisms and instant messaging malware.

2.1 E-mail Spamming Mechanisms

In this section, we first present the spam laundering mechanisms, and then briefly describe other commonly-used spamming approaches.

2.1.1 Spam Laundering Mechanisms

Spam laundering refers to the spamming process, in which only proxies are involved in origin disguise. The proxy refers to the application such as SOCKS [49] that simply performs “protocol translation” (i.e., rewrite IP addresses and port numbers) and forwards packets. Different from an e-mail relay, which first receives the whole message and then forwards it to the next mail server, an e-mail proxy requires that the connections on both sides of the proxy synchronize during the message transferring. More importantly, unlike an e-mail relay which inserts the information—

“Received From” that records the IP address of sender and the timestamp when the message is received—in front of the message header before relaying the message, an e-mail proxy does not record such trace information during protocol transformation. Thus, from a recipient’s perspective, the e-mail proxy, instead of the original sender, becomes the source of the message. It is this identity replacement that makes e-mail proxy a favorite choice for spammers.

Initially, spammers just seek open proxies on the Internet, which usually are misconfigured proxies allowing anyone to access their services. There are many Web sites and free software providing open proxy search function. However, once such misconfigurations are corrected by system administrators, spammers have to find other available “open” proxies. It is ideal for a spammer to own many “private” and stable proxies. Unsecured home PCs with broadband connections are good candidates for this purpose. Malicious software including specially-designed worms and viruses, such as SoBig and Bagle, has been used to hijack home PCs. Equipped with Trojan horse or backdoor programs, these compromised machines are available zombies. After proxy programs such as SOCKS or Wingate are installed, these zombies are ready to be used as spam proxies to pump out e-mail spam. Without serious performance degradation, most nonprofessional Windows users are not aware of the ongoing spamming. Recent research on the network-level behavior of spammers [72] also confirms that most sinked spam is originated from compromised Windows hosts.

To counter the soaring growth of spam volume, many ISPs have adopted the policy of blocking port 25 (SMTP [47] port), in which outbound e-mail from a subscriber must be relayed by the ISP-designated e-mail server. In other words, the ISP’s edge routers only forward the SMTP traffic from some designated IP addresses to the outside. However, spammers have easily evaded such simple SMTP port blocking mechanisms. The spam laundry is simple: having zombies send spam

messages to their ISP e-mail servers first. In February 2005, Spamhaus [91] reported that over the past few months a number of major ISPs had witnessed far more spam messages coming directly from the e-mail servers of other ISPs. This change in proxy-based spamming activity is mainly caused by the use of new stealthy spamware, which instructs the hijacked proxy (i.e., zombie) to send spam messages via the legitimate e-mail server of the proxy's ISP.

2.1.2 Other Spamming Approaches

The other commonly-used spamming approaches vary from dummy ISP spamming to more recent botnet spamming. We briefly summarize them as follows.

Act as a dummy ISP: Some professional spammers play this trick with ISPs to extend the duration of their spamming business. By purchasing a large amount of bandwidth from commercial ISPs and setting up a dummy ISP, these professional spammers pretend to have “users”, which seemingly need Internet access but in fact are used for spamming. If they are tracked for spamming, those spammers claim to their ISPs that the spam is sent by their nonexistent “customers”. A spammer achieves an extended spamming time by lying to one ISP, and later moving to another ISP. To evade anti-spam tracking and lawsuit, many professional spammers operate “offshore” by using servers in Asia and South America.

Spam through open-relay: To provide high reliability for e-mail delivery, SMTP [47] was designed to allow relaying. It means that some MTAs (Mail Transfer Agents) may help the originator MTA to transmit e-mail messages to the destination MTA, when the direct transmission from the originator to the destination is broken. Such a relaying service is unnecessary in current Internet environment and most MTAs have disabled the relay service for untrustworthy sources. However, due to misconfiguration or lack of experience, there are still many open-relays available

in the Internet [89].

Exploit CGI security flaws: Some insecure Web CGI (Common Gateway Interface) services, such as notorious FormMail.pl [80] that allows Internet users to send e-mail feedback from an HTML form, have been exploited by spammers to redirect e-mail to arbitrary addresses. This CGI-based e-mail redirection is appealing to spammers, since it can conceal the spam origin.

Hijack BGP routes and steal IP blocks: Some spammers are also Internet hackers. They hijack insecure BGP (Border Gateway Protocol) routers, pirate or fraudulently obtain some IP address allocations from an IP address assignment agency such as ARIN (American Registry for Internet Numbers), and use routing tricks to simulate faked networks, deceiving real ISPs into serving them connectivity for spamming. This spamming trick is also called “BGP spectrum agility” [72].

Spam through botnet: Recent studies have witnessed the wide use of botnets in spamming [17, 72] and phishing [105]. Using IRC (Internet Relay Chat) channels or other communication protocols, a bot controller (also a spammer) first distributes the spam address list and message content to all controlled bots. Then he sends a single command to bots, triggering the mailing engine installed on bots to pump spam. For a bot controller that is not directly involved in spamming, he may install spam proxies on bots and then lease his botnet to spammers for spam laundering.

2.2 Instant Messaging Malware

An instant messaging (IM) system is a real-time message delivery system that has dedicated servers for account management and message relay. IM systems can be categorized into either public IM systems or private IM systems (also called enterprise IM systems) by their targeted

users and usage environment. Public IM systems are open to everybody over the Internet while private IM systems are only accessible to a certain group of people (e.g., enterprise employees) inside a specific network. AOL Instant Messenger (AIM) and Windows Messenger series (MSN) illustrate the former while Reuters Messaging (RM) exemplifies the latter.

The IM malware studied in this dissertation refers to any malicious code that spreads through either public IM systems or private IM systems. For example, Opanki [46] (attacking AIM), Bropia [45] (attacking MSN), and Sohana [88] (attacking YIM) are typical examples of such malware. Although most of known IM malware spreads on popular public IM networks, enterprise IM systems such as Microsoft Office Live Communications Server [59] and IBM Lotus Same-time [41] can also be penetrated as these corporate IM services usually provide connectivity and interoperability with public IM services. In 2005, the outbreak of a variant of Kelvir worm even forced Reuters to shut down its IM service [38].

IM malware propagates mainly through two ways: malicious file transfer and malicious URL-embedded message. Malware infection is usually triggered by the victim's action such as clicking the accepted file or the received URL. IM malware could also spread without victim's involvement, for instance, by exploiting the vulnerabilities in IM clients. However, this type of spreading is rare.

The file transfer mechanism has been used since early 2000s. In this mechanism, IM malware propagates by initiating malicious file transfers to remote contacts. Malicious files are usually renamed to attract victims or to evade network filters. Once an unwary contact clicks the file, the malware is invoked and will attempt to infect more victims in the contact list (also called buddy list). To counter this type of malware spreading, some IM systems such as MSN forbid IM clients to transfer certain types of files such as *.pif* files. While the actual file transfer is normally carried out directly between two IM clients, the messages for transfer establishment

still go through IM server. Therefore, IM servers can easily detect the messages for establishing malicious file transfers and silently drop them to block malware propagation.

Nowadays malicious URL-embedded messages become much more popular than malicious file transfers for IM malware propagation. Instead of sending a file, IM malware sends a crafted text message containing a malicious URL to remote contacts. As soon as a victim clicks the link, either a malware binary is downloaded and executed or some malicious Web scripts run to exploit the vulnerabilities of the Web browser or other related applications. Compared to malicious file transfers, malicious URL messages have several advantages in propagation. First, malicious URL messages have more means to compromise a system. File downloading is just one of its attacking vectors. Second, malicious URLs can be used to collect victims' information by exploiting Web functionality. For instance, the URL sent by Kelvir.k [77] points to a php script and contains the contact's e-mail address. The e-mail address is harvested as soon as the URL is clicked. Last but not least, IM malware can play more social engineering tricks on URLs. For example, a malicious URL can be crafted to mimic the link on a reputable Web site [34]. The IM clients supporting HTML scripts also provide a playground for IM malware to fake URLs at their will. Those forged URLs appear normal but in fact point to malicious Web pages.

After infection, IM malware may take different actions for propagation. Many types of malware start spreading immediately after they compromise IM clients, while others wait until they receive instructions to spread. The latter usually install certain bot programs on compromised machines, through which the malware is controlled by the remote bot herder. IM malware might choose different targets in spreading. Some types of IM malware only hit online contacts, while other types also try offline users.

Although the threat of IM malware, especially the outbreak of zero-day IM malware, is on the

rise, network administrators still lack effective solutions to protect enterprise-like networks such as campus networks and corporate networks. Conventional protections using firewalls and anti-virus products are insufficient to defend against IM malware. Most of popular IM protocols are able to circumvent firewalls if their default ports are blocked. Signature-based anti-virus products cannot detect zero-day IM malware. Meanwhile, anomaly detection techniques, such as Norman Sandbox technology [64], may also be ineffective in catching evasive malware which behaves differently in the sandbox environment. Compared to malicious file transfers, malicious URL-embedded IM messages are even harder to be identified by anti-virus programs.

IM providers may take quick responses, for example, releasing patches and mandating client upgrade, to newly discovered vulnerabilities in their products. They may even proactively block potentially malicious file transfers. However, these filtering mechanisms still could be bypassed [78, 79]. Moreover, it is extremely hard for IM providers to protect against malicious URLs that exploit the vulnerabilities of Web browsers or other related applications [70]. While some protection schemes, such as CAPTCHA [57] and IM virus throttling [108], can enhance IM security, the incurred overhead and usability degradation could be significant, and thus prohibit IM providers from using them in near future.

Chapter 3

Thwarting E-mail Spam Laundering

E-mail spam proxies such as off-the-shelf SOCKS [49] and HTTP [29] proxies play an important role in the spam epidemic. Spammers launder e-mail spam through spam proxies to conceal their real identities and reduce spamming cost. The popularity of proxy-based spamming is mainly due to the anonymous characteristic of a proxy and the availability of a large number of spam proxies. The IP address of a spammer is obfuscated by a spam proxy during the protocol transformation, which hinders the tracking of real spam origins. According to Composite Blocking List (CBL) [23], which is a highly-trusted spam blacklist, the number of available spam proxies and bots in October 2009 was more than seven million. These numerous spam proxies facilitate the formation of e-mail spam laundering, by which a spammer has great flexibility to change spam paths and bypass anti-spam barriers. However, there is very little research done in detecting spam proxies. Probing is a common method used to verify the existence of spam proxies in practice. Probing works by scanning open ports on the spam hosts and examining whether or not e-mail can be sent through the open ports. Due to the wide deployment of firewalls and the use of scanning, both accuracy and efficiency of probing are poor.

In this chapter, we present a simple and effective mechanism, called *DBSpam*, which detects and blocks spam proxies' activities inside a customer network in a timely manner, and further traces the corresponding spam sources outside the network. DBSpam is designed to be placed at a network vantage point such as the edge router or gateway that connects a customer network to the Internet. The customer network could be a regional broadband (cable or DSL) customer network, a regional dialup network, or a campus network. DBSpam detects ongoing proxy-based spamming by monitoring bidirectional traffic. Due to the protocol semantics of SMTP (Simple Mail Transfer Protocol) [47] and timing causality, the behavior of proxy-based spamming demonstrates the unique characteristics of connection correlation and packet symmetry. Utilizing this distinctive spam laundering behavior, DBSpam can easily identify the suspicious TCP connections involved in spam laundering. Then, DBSpam can quickly single out the spam proxies, trace the spam sources behind them, and block the spam traffic.

This chapter is organized as follows. First, we present the unique behavior of proxy-based spamming and reveal the salient characteristic of packet symmetry in Section 3.1. Then, we detail the working mechanism of DBSpam in Section 3.2. We evaluate the effectiveness of DBSpam through the trace-based experiments in Section 3.3. We further discuss the robustness of DBSpam against potential evasions in Section 3.4. We survey related anti-spam techniques in Section 3.5. Finally, we summarize the contributions of this chapter in Section 3.6.

3.1 Proxy-based Spam Behavior

In this section, we delineate the distinct behavior of proxy-based spamming, which directly inspires the design of our detecting algorithm. Figure 3.1 depicts a typical scenario of proxy-based spamming in a customer network such as a Cox regional residential network. Although spammers

can conceal their real identities from destination MTAs by exploiting spam proxies, they cannot make the connection between a spam source and its proxy *invisible* to the edge router or gateway that sits in between. Here we assume that there is a network vantage point where we can monitor all the bidirectional traffic passing through the customer network, and the location of the gateway (or firewall) of the customer network (e.g. edge router R in Figure 3.1) that connects to the Internet is such a point.

3.1.1 Laundry Path of Proxy-based Spamming

As shown in Figure 3.1, there is a customer network N , in which spam proxies reside. Both spammer S and receiving MTA M are connected to customer network N via edge router R . S may be the original spam source or just another spam proxy (but it must be closer to the real spam source). M is the outside MTA.

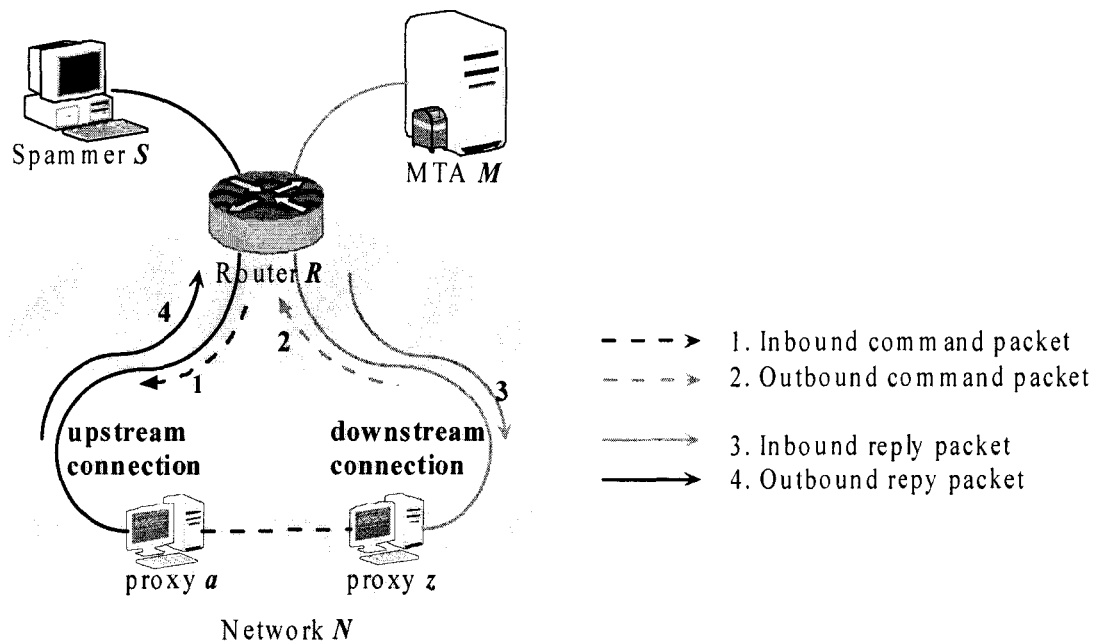


Figure 3.1: Scenario of proxy-based spamming

Note that for the customer network that has its own mail server(s) such as a campus (or an enterprise) network, the monitored network N may not be the whole network, but one of its protected sub-networks. Usually such campus/enterprise networks are divided into multiple sub-networks for security and management concerns. Their mail servers are placed in DMZ (DeMilitarized Zone) or a special sub-network that is separated from other sub-networks such as wireless, dormitory, or employee sub-networks. It is one of these loosely-managed sub-networks that becomes the monitored network N and the router/gateway connecting the sub-network N becomes the vantage point R . Thus, the assumption of exterior MTA M is valid even when the MTA is under the same administration domain as network N .

Inside monitored network N , S may use a single or multiple spam proxies. If multiple proxies are employed, they may either launder spam messages individually or be organized into one or multiple proxy chains, depending on the spammer's strategy. Without loss of generality, only one chain is shown in Figure 3.1. Spammer S usually communicates with spam proxies through SOCKS or HTTP. The spam message sent from S to a may even be encrypted. If it is a proxy chain, the spam message can be conveyed by different proxy protocols at different hops. For instance, SOCKS 4 is used between S and a , while HTTP is employed between a and z . However, none of these protocol variations and message content encryptions can change the fact: it is last-hop proxy z ¹ that does the protocol transformation and forwards the spam message to the MTA via SMTP.

We define the connection between spammer S and first-hop proxy a as the *upstream* connection, and define the connection between last-hop proxy z and MTA M as the *downstream* connection. The upstream and downstream connections plus the proxy chain form the spam laundry path, which is shown in Figure 3.1.

¹proxy z and proxy a are the same in the single proxy scenario.

3.1.2 Connection Correlation

There is a one-to-one mapping between the upstream and downstream connections along the spam laundry path. While this kind of connection mapping is common for proxy-based spamming, it is very unusual for normal e-mail transmission. In normal e-mail delivery, there is only one connection, that is, the connection between sender and receiving MTA. The existence of such connection correlation is a strong indication of spam laundering and provides valuable clue for spammer tracking. Here we assume that the downstream connection is an SMTP connection. For the upstream connection we have no restriction except that it should be a TCP connection. The packets in the upstream connection may be encrypted and even compressed.

The detection of such spam-proxy-related connection correlation is challenging due to the following three reasons. First, content-based approaches could be ineffective as spammers may use encryption to evade content examination. Second, because such a detection mechanism is usually deployed at network vantage points, the induced overhead should be affordable, which is critical to the success of its deployment. Third, since spam traffic is machine-driven and could be delayed by proxy at will, those timing-based correlation detection algorithms such as [113] may not work well in this environment.

3.1.3 Packet Symmetry

Figure 3.2 illustrates the detailed communication processes of spam laundering for both single proxy and proxy chain cases at the application layer, in which the message format is “PROTOCOL [content]”. For simplicity, P/P1/P2 stands for different application protocols, including SOCKS (v4 or v5), HTTP, etc. For SMTP, its packet content is in plain-text. But for application protocols P/P1/P2, their packet contents may be encrypted. Since the small delays induced by message pro-

cessing at end hosts and intermediate proxies have little effect upon the communication processes, for ease of presentation, we ignore them in Figure 3.2. The initial proxy handshaking process is also omitted as it has no effect on e-mail transactions. Without losing any generality, here we only show the shortest SMTP transaction process for the single-proxy case and parts of SMTP transaction process for the proxy-chain case.

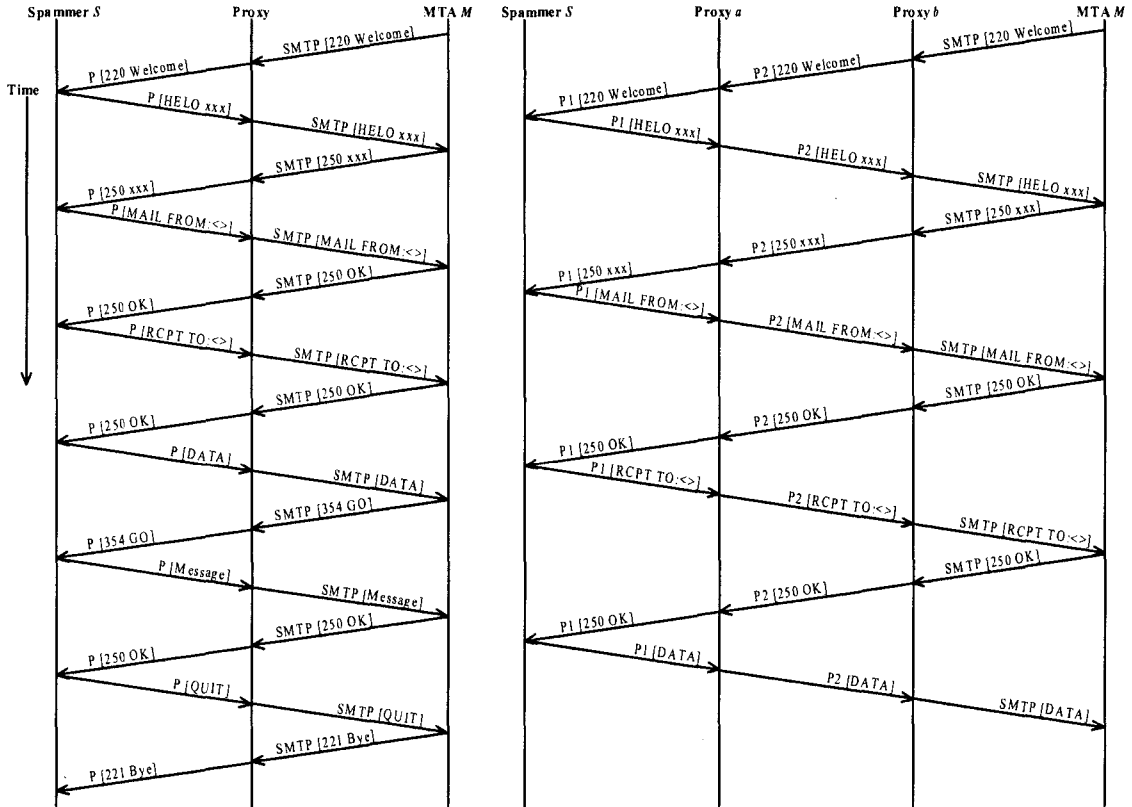


Figure 3.2: Time-line of spamming processes for single proxy (left) and proxy chain (right)

Due to protocol semantics, the process of proxy-based spamming is similar to that of an interactive communication. The appearance of one inbound SOCKS-encapsulated (or HTTP-encapsulated)² SMTP command message on the upstream connection will trigger the occurrence

²For the ease of presentation, we only use SOCKS in the rest of chapter, although HTTP can be used as well.

of one outbound SMTP command message on the downstream connection later. Similarly, for each inbound SMTP reply message on the downstream connection, later on there will be one corresponding outbound SOCKS-encapsulated reply message carried by TCP on the upstream connection. We term this communication pattern as *message symmetry*.

This message symmetry leads to the *packet symmetry* at the network layer with a few exceptions, in which the one-to-one packet³ mapping between the upstream and downstream connections may be violated. The exceptions can be caused by (1) packet fragmentation, (2) packet compression, (3) packet retransmission occurring along the laundry path. However, due to the fact that SMTP reply messages are very short (usually less than 300 bytes including packet header) and Path MTUs for most customer networks are above 500 bytes, the occurrence of (1) and (2) is very rare. Moreover, the packet retransmission problem can be easily resolved by checking TCP sequence numbers. In general, the packet symmetry between the inbound and outbound reply packets holds most of time.

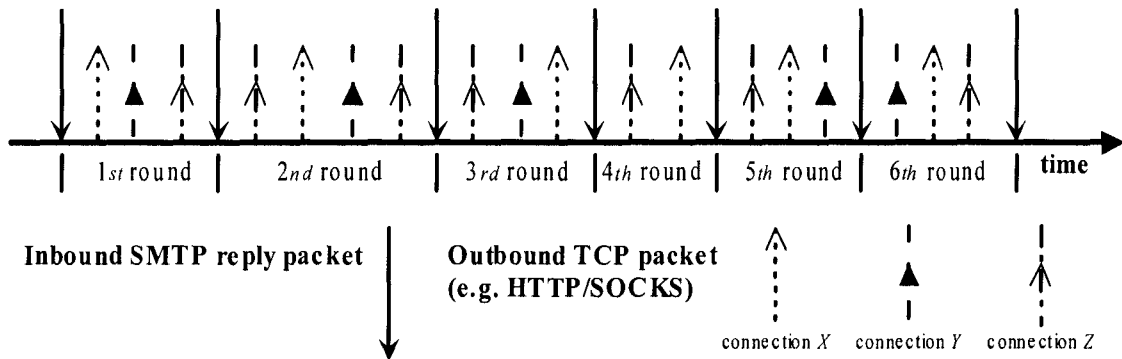


Figure 3.3: Example of reply round and TCP correlation

Such packet symmetry is exemplified in Figure 3.3, where the arrow with long solid line stands for the arrival of an inbound SMTP reply packet of the suspicious SMTP connection. In addition to

³TCP control packets such as SYN, ACK are not counted here.

the inbound SMTP connection, there are three outbound TCP connections X , Y , and Z , as shown in Figure 3.3. Three kinds of arrows with different dotted lines stand for the arrivals of outbound TCP packets belonging to these outbound TCP connections, respectively. The upward arrow indicates that the packet is leaving the monitored network, while the downward arrow indicates the packet is entering the network.

All of the inbound SMTP reply packets shown in Figure 3.3 belong to the same suspicious SMTP connection. We define a *reply round* as the time interval between the arrivals of two consecutive reply packets on an SMTP connection. Thus, the n_{th} reply round is the time interval between the arrival of the n_{th} reply packet and that of the $(n + 1)_{th}$ reply packet. Even for the simplified SMTP transaction, it has six reply rounds as shown in Figure 3.3. Within one reply round, the number of arrows with a specific dotted line indicates the number of outbound TCP packets of the corresponding TCP connection.

According to the one-to-one mapping of packet symmetry, each SMTP reply packet observed on the downstream SMTP connection should cause *one and only one* TCP packet appeared on the upstream connection. As Figure 3.3 shows, if one connection among X , Y , and Z is the suspicious upstream connection, one and only one outbound TCP packet must be observed from that connection in every reply round. Based on this rule, only TCP connection X meets this “one and only one” requirement and can be classified as the suspicious upstream connection with high probability. In the second reply round, more than one packets appear on connection Z ; and in the fourth round, no packet occurs on connection Y . Thus, we can easily filter out TCP connections Y and Z as normal background traffic. Note that the order of packet arrivals in a reply round does not affect the checking result of packet symmetry.

This *packet symmetry* is the key to distinguish the suspicious upstream and downstream con-

nections along the spam laundry path from normal background traffic. It simply captures the fundamental feature of chained interactive communications, and does not assume any specific time distribution of packet arrivals. We use this simple rule to detect the laundry path of proxy-based spamming, and the detection scheme is robust against any possible time perturbation induced by spammers. Note that the *one and only one* mapping of packet symmetry can be relaxed, which we will discuss in Section 3.4.

3.2 Working Mechanism of DBSpam

DBSpam consists of two major components: spam detection module and spam suppression module, in which the detection module is the core of DBSpam. To the best of our knowledge, so far there is no effective technique which can online detect both spam proxies and the corresponding spammers behind them. We envisage that DBSpam may achieve the following goals: (1) fast detection of spam laundering with high accuracy; (2) breaking spam laundering via throttling or blocking after detection; (3) support for spammer tracking and law enforcement; (4) support for spam message fingerprinting; and (5) support for global forensic analysis.

In essence, the detection module of DBSpam is a simple and efficient *connection correlation detection* algorithm to identify the laundry path of spam messages (i.e., the suspicious downstream and upstream connections) and the spam source⁴ that drives spamming behind the proxies.

3.2.1 Deployment of DBSpam

Like other network intrusion detection systems, DBSpam needs to be placed at a network vantage point that connects a customer network to the Internet, where it can monitor the bidirectional traffic

⁴Or just another spam proxy that is outside the customer network but at least one more step closer to the real source.

of the customer network. For a single-homed network, it is easy to locate such a network vantage point (an edge router or a firewall) and deploy DBSpam on it. For a multi-homed network, it may not be possible to locate a single network vantage point that can monitor all the bidirectional traffic passing through the customer network.

However, on one hand, many customer networks use multi-homing not for load-balance, but for reliability and fault-tolerance. Therefore, in case of the backup multi-homing, DBSpam works well if deployed at the primary ISP edge router. On the other hand, even in the load-balance multi-homing scenario, as long as the packets that belong to the same proxy chain go through the same ISP edge router or firewall, DBSpam still can work at different ISP edge routers or firewalls without coordination. Moreover, there are special network devices (e.g., Top Layer [6]) which can passively aggregate traffic from multiple network segments. By hooking up to such devices, DBSpam can still have the complete view of network traffic.

3.2.2 Design Choices and Overview

Our goal is to detect the spam laundry path promptly and accurately, once a proxy-based spamming activity occurs on the monitored network. We show in the previous section that packet symmetry is the inherent characteristic of proxy-based spamming behavior. Since legitimate messages are rarely delivered along the path illustrated in Figure 3.1, the possibility of a normal SMTP connection being consistently correlated with an unrelated TCP connection is very small in terms of packet symmetry. Hence, frequent observations of connection correlation is a strong indication of occurrence of spam laundering.

According to the packet symmetry rule, for the upstream TCP connection along a spam laun-

dry path, its outbound packet⁵ number in each reply round of the downstream SMTP connection is always one. For a normal TCP connection, however, this rule can only be satisfied with a very small probability. Thus, a simple and intuitive correlation detection method is to count the number of outbound packets observed on suspicious TCP connections in sequential reply rounds of an SMTP connection. Given the characteristic of successive arrival of observations, this correlation detection problem is well suited for the statistical method of *Sequential Probability Ratio Test* (SPRT) developed by Wald [102].

As a simple and powerful mathematical tool, SPRT has been used in many areas such as portscan detection [43] and wireless MAC protocol misbehavior detection [69]. Basically, an SPRT can be viewed as a one-dimensional random walk. The walk starts from a point between two boundaries and can go either upward or downward with different probabilities. With each arrival of observation, the walk makes one step in the direction determined by the result of observation. Once the walk first hits or crosses either the upper boundary or the lower boundary, it terminates and the corresponding hypothesis is selected. For SPRT, its actual false positive probability and false negative probability are bounded by predefined values. It has been proved that SPRT minimizes the average number of required observations to reach a decision among all sequential and nonsequential tests, which do not have larger error probabilities than SPRT.

We utilize the packet symmetry of SMTP reply packets to detect proxy-based spamming activity. Basically, we monitor the inbound SMTP traffic first, then apply the rule of packet symmetry for detecting the spam laundry path inside the customer network. In other words, DBSpam focuses on the clockwise reply packet flow as shown in Figure 3.1, instead of the counter-clockwise command packet flow, for connection correlation detection. The arrivals of inbound SMTP reply

⁵Here packets refer to nonretransmitted, nonzero-payload TCP packets.

packets, which delimit the reply rounds and drive the progress of connection correlation detection, become a self-setting clock of the detection algorithm. SPRT terminates by either selecting the hypothesis that upstream connection C_{tcp} is correlated with downstream connection C_{smtp} or choosing the opposite hypothesis.

There are two benefits of using SMTP reply messages to drive SPRT. First, as mentioned earlier, SMTP reply messages are very small, which minimizes the occurrence of packet fragmentation; and we can significantly increase the processing capacity of DBSpam by monitoring small packets only. Second, being either the spam target or the relay, the remote SMTP servers are usually very reliable; and the implementation and listening port of these servers strictly follow the SMTP protocol semantics. Thus, the packet symmetry rule always holds, and SMTP packets can be easily identified based on the port number of TCP header.

In the rest part of the section, we first briefly describe the basic concept of SPRT, then present the detection module of DBSpam, which include two phases: SPRT detection and noise reduction.

3.2.3 Sequential Probability Ratio Testing

Let $X_i, i = 1, 2, \dots$, be random variables representing the events observed sequentially. The SPRT for a simple hypothesis H_0 against a simple alternative H_1 has the following form:

$$\begin{aligned}
 \Lambda_n \geq B &\implies \text{accept } H_1 \text{ and terminate test,} \\
 \Lambda_n \leq A &\implies \text{accept } H_0 \text{ and terminate test,} \\
 A < \Lambda_n < B &\implies \text{conduct another observation,}
 \end{aligned} \tag{3.1}$$

where two constants or boundaries A and B satisfy $0 < A < B < \infty$, and Λ_n is the log-likelihood ratio defined as follows:

$$\Lambda_n = \lambda(X_1, \dots, X_n) = \ln \frac{\Pr(X_1, \dots, X_n | H_1)}{\Pr(X_1, \dots, X_n | H_0)}. \quad (3.2)$$

Assume X_1, \dots, X_n are independent and identically distributed (i.i.d.) Bernoulli random variables with

$$\Pr(X_i = 1 | \theta) = 1 - \Pr(X_i = 0 | \theta) = \theta, \quad i = 1, \dots, n. \quad (3.3)$$

Then

$$\Lambda_n = \ln \frac{\prod_1^n \Pr(X_i | H_1)}{\prod_1^n \Pr(X_i | H_0)} = \sum_1^n \ln \frac{\Pr(X_i | H_1)}{\Pr(X_i | H_0)} = \sum_1^n Z_i, \quad (3.4)$$

where $Z_i = \ln \frac{\Pr(X_i | H_1)}{\Pr(X_i | H_0)}$. Λ_n can be viewed as a random walk (or more properly a family of random walks⁶) with steps Z_i which proceeds until it first hits or crosses boundary A or B . Suppose the distributions for H_1 and H_0 are θ_1 and θ_0 , respectively. Λ_n moves up with step length $\ln \frac{\theta_1}{\theta_0}$ when $X_i = 1$, and goes down with step length $\ln \frac{1-\theta_1}{1-\theta_0}$ when $X_i = 0$.

In SPRT, we define two types of error

$$\alpha = \Pr(S_1 | H_0), \quad \beta = \Pr(S_0 | H_1),$$

where $\Pr(S_i | H_j)$ denotes the probability of selecting H_i but in fact H_j is true. If we call the selection of H_1 detection and the selection of H_0 normality, the event of $S_1 | H_0$ can be viewed as a false positive. So, α represents the false positive probability. Likewise, the event of $S_0 | H_1$ can be termed a false negative and β represents false negative probability.

⁶It is a family of random walks, since the distribution of the steps depends on which hypothesis is true.

Let α^* and β^* be user-desired false positive and false negative probabilities, respectively.

According to (3.1), we can derive⁷ the *Wald boundaries* as follows:

$$A = \ln \frac{\beta^*}{1 - \alpha^*}, \quad B = \ln \frac{1 - \beta^*}{\alpha^*}, \quad (3.5)$$

and the derived relationships between actual error probabilities and user-desired error probabilities are:

$$\alpha \leq \frac{\alpha^*}{1 - \beta^*}, \quad \beta \leq \frac{\beta^*}{1 - \alpha^*}, \quad (3.6)$$

$$\alpha + \beta \leq \alpha^* + \beta^*. \quad (3.7)$$

Inequality (3.6) suggests that the actual error probabilities α and β can only be slightly larger than their expected values α^* and β^* . For example, if the desired α^* and β^* are both 0.01, then their actual values α and β will be no greater than 0.0101. Inequality (3.7) can be interpreted as that the sum of actual error probabilities is bounded by the sum of their desired values.

According to Wald's theory, $E[N] = E[\Lambda_N]/E[Z_i]$. Here N denotes the number of observations when SPRT terminates. Suppose hypothesis H_1 is true and Bernoulli variable X_i has distribution θ_1 which implies that Λ_n steps up with probability θ_1 or goes down with probability $1 - \theta_1$, we have

$$E[Z_i|H_1] = \theta_1 \ln \frac{\theta_1}{\theta_0} + (1 - \theta_1) \ln \frac{1 - \theta_1}{1 - \theta_0}. \quad (3.8)$$

If the user-desired false negative probability of the test is β^* , then the true positive probability is $1 - \beta^*$ and

$$\begin{aligned} E[\Lambda_N|H_1] &= \beta^* A + (1 - \beta^*) B \\ &= \beta^* \ln \frac{\beta^*}{1 - \alpha^*} + (1 - \beta^*) \ln \frac{1 - \beta^*}{\alpha^*}. \end{aligned} \quad (3.9)$$

⁷The derivations of (3.5), (3.6), and (3.7) are omitted here. See [43, 102] for details.

With (3.8) and (3.9), we have

$$E[N|H_1] = \frac{\beta^* \ln \frac{\beta^*}{1-\alpha^*} + (1-\beta^*) \ln \frac{1-\beta^*}{\alpha^*}}{\theta_1 \ln \frac{\theta_1}{1-\theta_1} + (1-\theta_1) \ln \frac{1-\theta_1}{\theta_1}}. \quad (3.10)$$

Likewise, we can derive

$$E[N|H_0] = \frac{(1-\alpha^*) \ln \frac{\beta^*}{1-\alpha^*} + \alpha^* \ln \frac{1-\beta^*}{\alpha^*}}{\theta_0 \ln \frac{\theta_1}{1-\theta_1} + (1-\theta_0) \ln \frac{1-\theta_1}{1-\theta_0}}. \quad (3.11)$$

Apparently the average observation number $E[N]$ of SPRT is determined by four parameters: pre-defined error probabilities α^* , β^* and distribution parameters θ_0 and θ_1 . The determination of these values and their effects on $E[N]$ will be discussed with our correlation detection algorithm in the following.

3.2.4 SPRT Detection Algorithm

According to the principle of packet symmetry, within each reply round, there must be one and only one outbound TCP packet appearing on the corresponding upstream connection. By contrast, those connections that have none or more than one TCP packet can be classified as innocent connections. Within the framework of SPRT, this correlation detection problem can be easily transformed into an SPRT, in which we test the hypothesis H_1 that C_{tcp} is correlated with C_{smtp} against the hypothesis H_0 that the two connections are uncorrelated by counting the number of TCP packets appearing on C_{tcp} in each reply round of C_{smtp} .

If we use a Bernoulli random variable X_i to represent the observation result on C_{tcp} in the i -th reply round of C_{smtp} and assume that these variables in different rounds are i.i.d., we have the following distribution:

$$\Pr(X_i|H_1) = \begin{cases} \theta_1 & \text{if one outbound TCP packet observed} \\ 1 - \theta_1 & \text{otherwise} \end{cases}$$

$$\Pr(X_i|H_0) = \begin{cases} \theta_0 & \text{if one outbound TCP packet observed} \\ 1 - \theta_0 & \text{otherwise} \end{cases}$$

Algorithm 1 describes the procedure of detecting connection correlation based on SPRT. The values of four parameters A, B, θ_0, θ_1 are specified beforehand. To identify if C_{tcp} and C_{smtp} are correlated, at the end of each reply round of C_{smtp} , the number of the outbound packets observed on C_{tcp} is counted. If the number is 1, Λ is incremented by $\ln \frac{\theta_1}{\theta_0}$; otherwise, it is incremented by $\ln \frac{1-\theta_1}{1-\theta_0}$. Then, the updated Λ is compared with A and B . If Λ is either no greater than A or no smaller than B , the detection terminates and the corresponding hypothesis is selected. Otherwise, the test continues. However, the detection still terminates if either C_{tcp} or C_{smtp} is closed before a hypothesis is derived. In this case, C_{tcp} and C_{smtp} are deemed uncorrelated.

For proxy-based spamming, given that packet symmetry holds most of time, the major reason that correlation cannot be detected is mainly attributed to the packet misses by the monitoring system. For example, when the traffic volume exceeds the capacity that the monitoring system can handle, packets may be dropped by the monitoring system. If the packet conveying an SMTP reply message is dropped on either the downstream connection or the upstream connection, the correlation detection will fail in this reply round. So we can use packet miss rate to estimate the probability of a proxy connection being correlated when spamming occurs, that is, θ_1 . From the conservative perspective, we take 0.01 as the packet miss rate which in fact is fairly high⁸ considering only small packets (say less than 300 bytes) need attention and only packet header information is required for detection algorithm. So θ_1 is 0.99 in this case.

To estimate θ_0 , we employ the mathematical model given in [20]. We assume that the unidirectional packet arrivals of a normal TCP connection can be modeled as a nonhomogeneous Poisson process, which can be approximated by a sequence of Poisson processes with varying rates, and

⁸In practice, the miss rate is usually below 0.005 in our campus network.

Algorithm 1 Detect-Correlation

```
1: Input:  $C_{tcp}, C_{smtp}$ 

2: Parameters:  $A, B, \theta_0, \theta_1$ 

3: Output:  $C_{tcp}$  is correlated with  $C_{smtp}$  or not

4: repeat

5:   for each reply round of  $C_{smtp}$  do

6:     if # of outbound packets on  $C_{tcp}$  is 1 then

7:        $\Lambda_n \leftarrow \Lambda_{n-1} + \ln \frac{\theta_1}{\theta_0}$ 

8:     else

9:        $\Lambda_n \leftarrow \Lambda_{n-1} + \ln \frac{1-\theta_1}{1-\theta_0}$ 

10:    end if

11:    if  $\Lambda_n \geq B$  then

12:       $C_{tcp}$  is correlated with  $C_{smtp}$  and the test stops

13:    else if  $\Lambda_n \leq A$  then

14:       $C_{tcp}$  is not correlated with  $C_{smtp}$  and the test stops

15:    else

16:      wait for observation in next reply round

17:    end if

18:  end for

19: until either  $C_{tcp}$  or  $C_{smtp}$  is closed
```

over varying time periods that could be arbitrarily small. For example, let $M(t)$ denote the number of packets sent in an outbound TCP connection during time interval t . Process $\{M(t), t \geq 0\}$ can be represented by a sequence of Poisson processes $(\lambda_1, \Delta t_1), (\lambda_2, \Delta t_2), \dots$, where $t = \Delta t_1 + \Delta t_2 + \dots$.

The advantage of this model is that it can approximate almost any distribution. More importantly, the number of packets observed during any given time interval T , can be represented by a Poisson process M with a single rate $\hat{\lambda}_T$. Here $\hat{\lambda}_T$ is the weighted mean of the rates of all the Poisson processes during T .

With this model, we can easily compute the probability of one and only one packet sent in a reply round if T denotes the duration of a reply round. From

$$\Pr(M = i) = e^{-(\hat{\lambda}_T T)} \frac{(\hat{\lambda}_T T)^i}{i!}, \quad (3.12)$$

we have

$$\Pr(M = 1) = e^{-(\hat{\lambda}_T T)} (\hat{\lambda}_T T) \leq e^{-1}. \quad (3.13)$$

In (3.13) $\Pr(M = 1)$ reaches its maximum value e^{-1} when $\hat{\lambda}_T T = 1$. Although this is a theoretical derivative, we find that it is valid on almost all of the evaluated traces. Thus, we set $\theta_0 = e^{-1}$.

If we choose 0.005 for false positive probability α^* and 0.01 for false negative probability β^* , with $\theta_0 = e^{-1}$ and $\theta_1 = 0.99$, $E[N|H_1]$ is 5.5 and $E[N|H_0]$ is 2.02, respectively. Figure 3.4 shows how $E[N|H_1]$ varies with the changes of α^* and θ_0 , when β^* and θ_1 are fixed. In general, $E[N|H_1]$ increases when θ_0 gets bigger or α^* gets smaller. Intuitively, this prolonged random walk is a natural result of smaller step length $\ln \frac{\theta_1}{\theta_0}$ or enlarged distance $\ln \frac{1-\beta^*}{\alpha^*}$ for the walk towards the upper threshold.

From the perspective of anomaly detection, it is desirable that error probabilities, especially the false positive probability, can be as low as possible. In the framework of SPRT, this implies that $E[N|H_1]$ goes up, that is, the average detection time is prolonged. However, given that not all SMTP transactions (the shortest one has only 6 reply rounds) can be longer enough to make the SPRT reach a decision when α is too small, a tradeoff between lowering false positive and

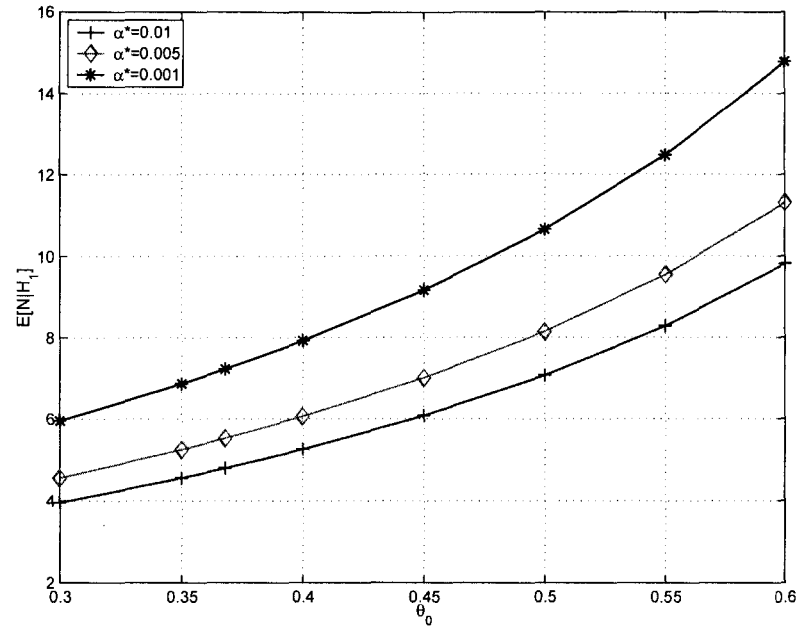


Figure 3.4: $E[N|H_1]$ vs. θ_0 and α^* ($\theta_1 = 0.99, \beta^* = 0.01$)

false negative has to be made. In DBSpam, we set $\alpha^* = 0.005$ so that even the shortest spam transactions can be captured.

3.2.5 Noise Reduction

To further lower the false positives of SPRT, we introduce a simple and effective noise reduction technique in DBSpam. In a series of correlation tests, we define the active spam sources and proxies that are prone to be identified many times as signals, and define those innocent IP addresses that may be accidentally captured as noises. We utilize the dichotomy between signal and noise to distinguish spam sources and proxies from innocent end hosts. We call this procedure *noise reduction*. The noise reduction are executed in two steps: first, we maintain a set S_i of external IP addresses that appear in the correlation results for each time window Δ ; second, in the consecutive M time windows, we single out the external IP addresses, which appear no fewer than K times, as

the spam sources and the corresponding proxy addresses as the spam proxies.

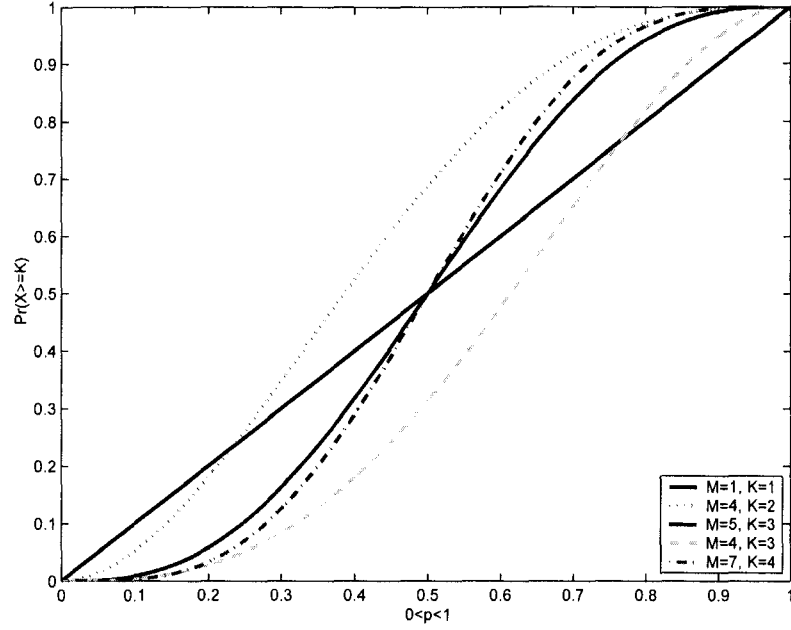


Figure 3.5: $\Pr(X \geq K)$ vs. p and (M, K)

The time window Δ is determined by the lower-bound of spamming rate ν (in replies/s) and the number of reply rounds N :

$$\Delta \geq N/\nu. \quad (3.14)$$

Hence, a spammer sending spam faster than ν must appear in S_i at least once in each time window Δ . Assume that the appearance of an IP address in S_i is independent, with a constant probability p . Then, the number of occurrences of the IP address among M time windows follows the *binomial* distribution.

$$\Pr(X = i) = \binom{M}{i} p^i (1-p)^{M-i}. \quad (3.15)$$

The probability of having no fewer than K occurrences in the binomial distribution is:

$$\Pr(X \geq K) = \sum_{i=K}^M \binom{M}{i} p^i (1-p)^{M-i}. \quad (3.16)$$

Figure 3.5 illustrates the dynamics of $\Pr(X \geq K)$ with the variation of probability p for several predetermined tuples of (M, K) . The diagonal line shows the case of tuple $(M = 1, K = 1)$, in which $\Pr(X \geq K)$ is equal to p . Clearly, if p is smaller than 0.2, all other curves are below this diagonal line, indicating that their values of $\Pr(X \geq K)$ are smaller than that of tuple $(M = 1, K = 1)$. In contrast, if p is larger than 0.8, these curves are above the diagonal line, indicating that their values of $\Pr(X \geq K)$ are larger than that of tuple $(M = 1, K = 1)$.

The value of p for an innocent address depends on the false positive rate of the correlation detection, which should be closer to zero than one. The left part of Figure 3.5 illustrates the noise reduction can further lower the chance of an innocent address being misclassified as a spam source. On the other hand, the value of p for a spam source is related to the complementary of the false negative rate of the correlation detection, which should be closer to one than zero as shown in the right part of Figure 3.5. This indicates that noise reduction increases the probability of a spam source being identified as well. Therefore, both false positives and false negatives are reduced after noise reduction. Figure 3.5 shows that when M is fixed, the probability $\Pr(X \geq K)$ goes smaller with bigger K . For example, $\Pr(X \geq 3|M = 4)$ is much smaller than $\Pr(X \geq 2|M = 4)$. Moreover, the noise reduction algorithm works very well even with very small M and K . For example, with $(M = 4, K = 3)$, pre-noise-reduction false positive rate, which is 0.1, can be significantly lowered to 0.0037 after noise reduction. These two rules of thumb may guide the selection of (M, K) in practice. We will further discuss the parameter setup of Δ , M and K , and demonstrate the effectiveness of the noise reduction technique in Section 3.3.3.

3.3 System Evaluation

We implemented a prototype of DBSpam using *libpcap* on Linux. Due to access limitation, we cannot deploy our prototype in an ISP network environment to evaluate its online performance. Alternatively, we collected traces from a middle-sized campus network and conducted a series of trace-based experiments to validate the efficacy of DBSpam.

By replaying the collected traces with our prototype, we attempt to answer the following questions: (1) how fast DBSpam can detect spam laundering; (2) how accurate the detection result of DBSpam is; (3) how many system resources DBSpam consumes.

3.3.1 Data Collection

The campus network is connected to the Internet via an OC-3 data link. A Snort-based NIDS [75] is deployed on the edge router of the campus network to block any suspicious proxy traffic (e.g. SOCKS and HTTP) via signature checking. All outgoing e-mail messages must go through the main e-mail server and secure authentication is enforced.

This well-protected campus network provides an ideal platform to assess the false positive ratio of DBSpam on normal network traffic. According to the IT department, proxy-based spamming activities on this campus network are very rare. To evaluate the detection time and accuracy of DBSpam on spam laundering, we generate “spam” traffic, including both plain-text and encrypted proxy traffic, with the cooperation of the IT department. Although the monitoring systems of IT can detect plain-text proxy traffic by checking content, our encrypted proxy traffic successfully evades their detection.

The generated spamming scenario is similar to the one shown in Figure 3.1. The campus network plays the role of network N . We use two home PCs outside the campus network, which

are located in two different ISP broadband networks, to emulate two spam sources. The spam sink (MTA M in Figure 3.1) is located in the dark net of the campus network. The dark net is a special subnet that directly links to the edge router and is used to dump all malicious traffic. One SOCKS proxy and one HTTP proxy running in two different subnets of the campus network form a proxy chain. We use a common spamming tool and *sockschain*⁹ to emulate proxy-chain spamming. The spam messages are sent from the two home PCs, through the proxy chain and destined to the spam sink. The data collection point is just before the edge router and can see all the traffic passing through the edge router. We use *tcpdump* to capture all small bidirectional TCP packets with the *snaplen* set to 75 bytes.

Table 3.1: Trace information

trace	duration (second)	packets	average pkt/sec	size (MB)	pkt miss rate	threads/ spammer
S-1-A	770	3,872,550	5,029	295	< 0.001	1
S-1-B	674	4,178,567	6,200	318	< 0.001	3
S-1-C	756	4,509,336	5,965	343	< 0.001	1
S-2-A	654	12,036,413	18,404	931	0.008	1
S-2-B	1,385	26,422,563	19,078	2,044	0.005	3
S-2-C	1,398	26,172,898	18,722	2,018	0.005	1
N-1	5,116	24,434,518	4,776	1,851	<0.001	-
N-2	14,944	297,733,228	19,923	22,950	0.006	-

We collected multiple traces of normal and spam traffic in two different months. The detailed

⁹Both are binary Windows programs so that we cannot modify any code.

information of the traces is listed in Table 3.1, and additional explanations are given below. First, we only captured small TCP packets with packet length less than 300 bytes as DBSpam only utilizes the SMTP reply messages for detection, which are usually conveyed by TCP packets with length less than 300 bytes. Second, we collected two kinds of traces to evaluate the performance of DBSpam, one with generated spam traffic and the other without generated spam traffic. All traces include the normal background SMTP traffic passing through the campus network. The name of a trace follows the format “{S|N}-{1|2}-{A|B|C}”. S (N) indicates that the trace has Spam (No spam) traffic. 1 (2) refers to the different month of trace collection. A (B, C) is only for spam traces and stands for different spam scenario. Third, in order to validate DBSpam for detecting both plain-text and encrypted spam traffic, we injected encrypted and compressed spam traffic through SSH tunneling into traces S-*-C (* is either 1 or 2), and injected plain-text spam traffic into S-*-A and S-*-B. Fourth, a multi-threaded spamming technique was used in S-*-B to validate the efficacy of DBSpam in a multi-threaded spamming scenario. The N-threaded spamming means up to N upstream connections may be issued simultaneously from the spam source to a proxy for spam laundering.

3.3.2 Detection Time

The overall detection time of DBSpam is determined by SPRT detection time, the noise-reduction time window Δ , and the number of consecutive windows M . Among these three factors, SPRT detection time is the fundamental one, which bounds the value of time window Δ . In the following, we focus on the estimation of SPRT detection time.

We evaluate SPRT detection time from two perspectives: the number of observations needed to reach a decision and the actual time spent by SPRT.

Table 3.2: Distribution of $N|H_1$

Trace	$N = 6$	$N = 11$	$N \geq 16$
S-1-A	970 (100%)	0	0
S-1-B	5019 (96.9%)	139 (2.7%)	21 (0.4%)
S-1-C	2245 (92.8%)	169 (7.0%)	6 (0.2%)
S-2-A	433 (99.1%)	3 (0.7%)	1 (0.2%)
S-2-B	4298 (94.7%)	198 (4.4%)	40 (0.9%)
S-2-C	1758 (98.9%)	16 (1.0%)	3 (0.1%)

Number of Observations N : The theoretical average number of observations under spam hypothesis ($E[N|H_1]$) and nonspam hypothesis ($E[N|H_0]$) can be easily computed based on Equations (3.10) and (3.11). In our evaluation, they are rounded to 6 and 3, respectively, with $\alpha^* = 0.005$, $\beta^* = 0.01$, $\theta_0 = e^{-1}$, and $\theta_1 = 0.99$. Table 3.2 shows the distribution of $N|H_1$ in six spam traces. The results clearly demonstrate the dominance of ($N = 6$) in all traces. The comparatively low percentage of ($N = 6$) in trace S-1-C is mainly caused by the abnormally high packet-miss-rate of the spam traffic but not the whole traffic. Note that due to the characteristics of SPRT, the detection of connection correlation (H_1) can only be reached after certain number of observations, such as 6 and 11.

Figure 3.6 shows the distribution of $N|H_0$ for nonspam traces N-1 and N-2. The curves indicate that SPRT can filter out at least 95% of normal connections within four observations. The distributions of $N|H_0$ for spam traces are similar to those for nonspam traces.

Actual Detection Time of SPRT: After recording the start and end points for each SPRT on six spam traces, we derive all the detection time in these traces and draw the CDFs (cumulative

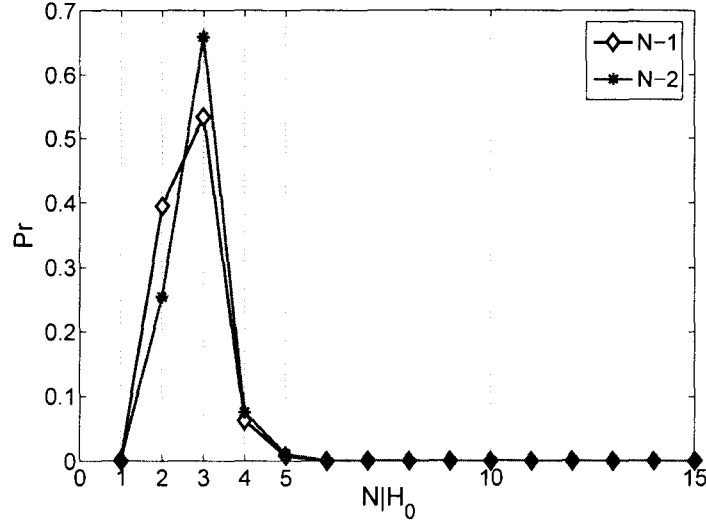


Figure 3.6: Distribution of $N|H_0$

distribution functions) in Figure 3.7. The detection time is approximated by ceiling for CDF drawing, e.g., 1.2s is ceiled to 2s. We classify the results from six traces into two groups: “S-1” and “S-2”, since the results in each group are very similar. As shown in Figure 3.7, 95% detections are made within 5 seconds. Note that the actual detection time is roughly the duration of 6 reply rounds of SMTP connection, since the computation overhead of SPRT is negligible. The curve difference between “S-1” and “S-2” is due to the inferior link quality in “S-2” experiments.

3.3.3 Detection Accuracy

Since the detection module of DBSpam has two phases—SPRT detection and noise reduction, we first evaluate the false positive and false negative of SPRT detection, and then present the overall detection accuracy of DBSpam after noise reduction.

(1) Accuracy of SPRT

False Positives: The left part of Table 3.3 shows the false positives of SPRT in different traces.

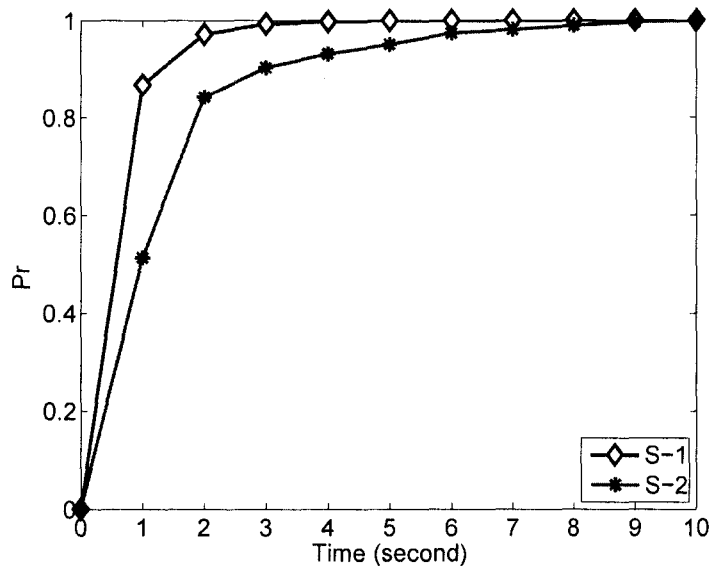


Figure 3.7: CDF of detection time for SPRT

The “detection” column is the total number of correlations reported by SPRT, and “True Positives (TP)” and “False Positives (FP)” columns list the outcome of detections. The “True Negatives (TN)” column lists the number of tests on normal connections that are correctly identified. According to the definition of false positive probability $\alpha = \frac{FPs}{FPs+TNs}$, the probabilities in all traces are well below 0.0002, indicating that the false positive probability of SPRT is fairly small in practice.

False Negatives: We estimate the false negatives by counting the number of proxy connections that are missed by SPRT, and compute the ratio of missed spam connections, which are shown in the right part of Table 3.3. The false negatives of SPRT are attributed to the missed packets in the spam traces. The three spam traces S-2-A/B/C contain both long SMTP connections (no less than ten reply rounds) and short SMTP connections (six reply rounds). More than half of the total connections are short SMTP connections. For those short spam connections with only six reply

Table 3.3: False positives and false negatives of SPRT

Trace	Detection	TPs	FPs	TNs	FPs/ (FPs+TNs)	Spam Conns	Missed Conns	Miss Ratio
S-1-A	970	966	4	290,889	1.4e-5	958	8	0.008
S-1-B	5,179	5,108	71	1,156,085	6.1e-5	570	2	0.004
S-1-C	2,420	2,369	51	596,979	8.5e-5	324	0	0
S-2-A	437	320	117	1,634,307	7.2e-5	329	6	0.018
S-2-B	4,536	3,510	1,026	8,895,993	1.2e-4	1,351	27	0.020
S-2-C	1,777	1,558	219	4,266,100	5.1e-5	969	13	0.013
N-1	66	-	66	687,390	9.6e-5	-	-	-
N-2	2,368	-	2,368	15,941,150	1.5e-4	-	-	-

*TP: True Positive, FP: False Positive, TN: True Negative

rounds, if any packet on either the upstream connection or the downstream connection is missed in the trace, SPRT cannot reach a decision, leading to a false negative. A simple estimation shows the feasibility of the missing ratio of spam connections. For simplicity, we assume that the packet miss rate p is constant through the trace. Then, the probability of one packet missing in six reply rounds is $\binom{12}{1}p(1-p)^{11}$. If $p = 0.005$ (the packet miss rate of traces S-2-B/C), the probability is around 0.057, which is more than the miss ratio as shown in Table 3.3.

(2) DBSpam Accuracy after Noise Reduction

To investigate the efficacy of noise-reduction, we first need to determine the value of time window Δ . Figure 3.7 shows that over 80% of all SPRTs on spam traces terminate within 2 seconds. So, we set the time window Δ to 2 seconds. For (M, K) , we test several combinations and the final

detection results are shown in Table 3.4, where the data format is “number of FP/number of overall detections”. From the table, we can see that noise reduction eliminates the majority of false positives of SPRT, due to the fact that most of wrongly-classified correlations only occur sporadically. The false positive number of DBSpam approaches zero, when (1) M and K are relatively large and (2) the gap between M and K is small. Such dynamics of false positive reduction fits well with the analysis in Section 3.2.5. For our traces, any combination with 4/5 for M and 3/4 for K can achieve fairly high accuracy. Of course, the high detection accuracy is achieved at the cost of lowering detection sensitivity. It always exists a tradeoff between accuracy and sensitivity in network anomaly detection. However, even when the time window Δ is set to 2 seconds and M is set to 5, the overall delay of DBSpam detection is just 10 seconds but with much higher accuracy.

Currently most false positives of DBSpam are induced by P2P applications. The capacity of spawning thousands of connections in a second and the behavior of periodic PING/PONG communications make P2P applications have a much higher probability of being correlated than any other applications. Due to the hog overwhelming proportion in bandwidth consumption, many ISPs and university networks in US have restricted the maximal connections that P2P applications can establish, which helps reduce the false positives of DBSpam.

3.3.4 Resource Consumption

According to Table 3.1, the arrival rate of small TCP packets at the edge router can reach around 20,000 packets per second (pps), at which DBSpam must be able to handle. Current high-end PCs can meet this requirement without much difficulty. Using a Dell Precision 360 machine with Pentium-4 3GHz CPU and 512MB memory, we run the prototype of DBSpam on each trace multiple times. We use *time* and *ps* to measure the CPU and memory usage. The results are listed

Table 3.4: Overall false positives of DBSpam ($\Delta = 2s$)

Trace	(M, K)			
	(3, 2)	(4, 3)	(5, 3)	(5, 4)
S-1-A	0/188	0/138	0/124	0/110
S-1-B	0/162	0/126	0/103	0/103
S-1-C	0/194	0/150	0/124	0/123
S-2-A	0/65	0/36	0/52	0/27
S-2-B	13/335	3/243	4/216	0/186
S-2-C	0/193	0/124	0/135	0/94
N-1	0/0	0/0	0/0	0/0
N-2	7/7	1/1	2/2	0/0

*Data Format: # of false positives / # of total detections

in Table 3.5. The average packet processing rate of DBSpam is computed by dividing the total packet number of the trace over the processing time (“CPU Time”). The processing rates clearly demonstrate the capability of DBSpam working at high-speed networks. Even in the worst case, DBSpam still can handle 241,965 pps, which is over 10 times more than the required processing speed.

Memory consumption of DBSpam is mainly determined by two factors: the number of active SMTP connections and the number of outbound TCP connections during each SMTP reply round. So, the peak memory consumption is not necessarily determined by the network traffic volume. As DBSpam only needs to maintain very few states, and only a very small portion (false positive probability) of connections need to maintain states for relatively long time (lifespan of SMTP con-

Table 3.5: Resource consumption

Trace	CPU Util	CPU Time	pps	Peak Mem
S-1-A	36.3%	9.0s	430,283	2.2MB
S-1-B	37.7%	9.8s	426,384	1.6MB
S-1-C	24.0%	9.3s	484,875	1.2MB
S-2-A	58.0%	36.8s	327,076	11.9MB
S-2-B	84.3%	109.2s	241,965	10.5MB
S-2-C	57.1%	78.6s	332,989	2.8MB
N-1	21.7%	51.1s	478,171	5.6MB
N-2	32.1%	789.9s	376,925	8.4MB

nections), the overall memory consumption should not be a problem. Also note that the memory management of our prototype is quite naive since our focus is mainly on the correctness, not on the performance.

3.3.5 Suppressing Spam Activities

Once the spam proxies and the spam sources behind them are identified, it is straightforward to suppress the spam activities inside the customer network. Two commonly-used approaches to suppressing proxy-based spam activities are rate-limit throttling and blocking.

We suggest blocking the inbound TCP traffic from the spam source to its abused proxies. In general, the spam source is highly likely a compromised machine or the end host where a spammer resides. It is rare that frequent innocent communications exist between a spam source and its proxies. Therefore, the collateral damage of blocking traffic from these identified spam

sources should be minor.

On the other hand, there may exist legitimate e-mail traffic between a spam proxy and the MTA as a legitimate user residing in the proxy machine may also send e-mail. To minimize the collateral damage, we conduct rate-limit throttling on the outbound SMTP traffic from spam proxies, instead of simple blocking. The setting of rate-limit is based on the normal e-mail traffic behavior between a nonspam client and the MTA, and can be tuned by network administrators.

To evaluate the efficacy of DBSpam on spam suppression, we activate the suppression module of DBSpam and simulate the spam suppression based on the collected traces. We use two machines for evaluating spam suppression, one for traffic generator and the other for traffic sink. We use *tcpreplay* [97] to inject traffic on the wire by replaying traces on the traffic generator, and then have DBSpam to detect and suppress spam activities on the traffic sink. The traffic sink simulates the edge gateway in a real environment.

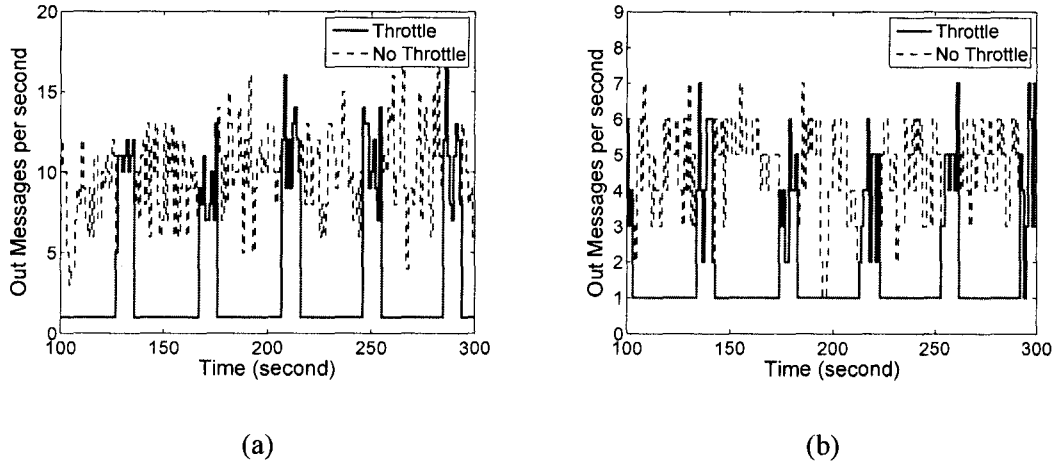


Figure 3.8: Comparison of number of messages sent out before and after throttling

We first examine DBSpam in spam throttling. We set the maximal mail sending rate as one message per second and throttling duration as 30 seconds. The suppression module silently drops

the excessive messages. Here we record message numbers by counting the number of “RCPT” commands appeared between “MAIL” and “DATA” commands in a transferring transaction. The mail transactions with multiple “RCPT” commands are delayed to meet the threshold of maximal sending rate. The parameters Δ , M , and K of the detection module are set to 2s, 4, and 3, respectively. After the detection module fires an alarm, the suppression modules is activated to throttle the spam proxy in the downstream connection of the laundry path, which lasts for the predefined time (i.e., 30s). Figure 3.8 shows the experimental results of DBSpam in throttling spam activities. Figure 3.8(a) shows an excerpt (from 100s to 300s in trace time) of the throttling result in trace S-1-B, and Figure 3.8(b) shows the corresponding result in trace S-2-C. The dynamics of spam message rates with and without throttling are shown as the red solid line and the blue dashed line, respectively. It is evident that as suppression is turned on, the spam sending rate is immediately dropped and limited to 1 message/second for next 30 seconds. The alternation of detection phase and suppression phase is also clearly shown in Figure 3.8.

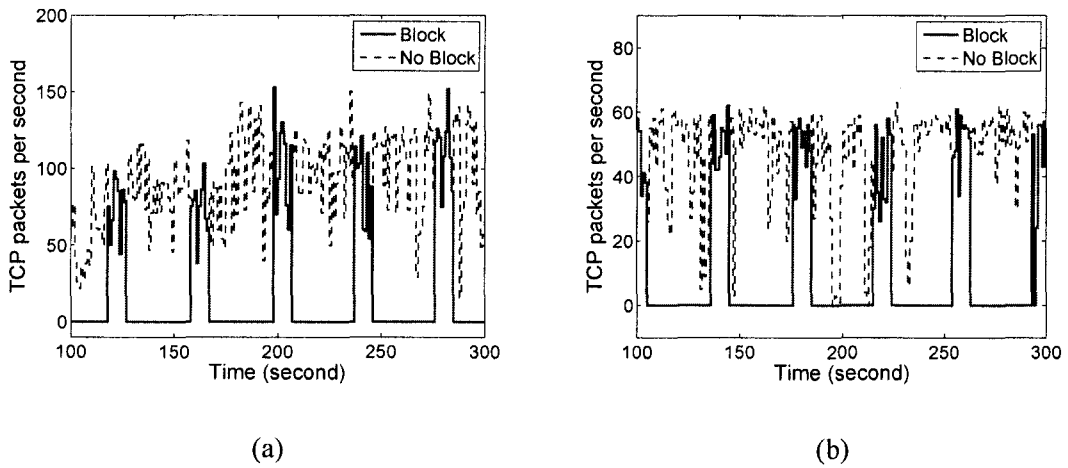


Figure 3.9: Comparison of TCP packet numbers before and after blocking

Then we test DBSpam in blocking TCP traffic from detected spam sources. The blocking

technique is quite simple, just dropping TCP packets from the flagged IP addresses. We use the same experimental setup for the blocking test as that for the throttling test, that is, the same parameter setting for detection module and 30 seconds for blocking duration. Figure 3.9 illustrates the dynamics of the TCP traffic from a specific spam source with and without blocking. Figure 3.9(a) is for trace S-1-B, and Figure 3.9(b) is for trace S-2-C. Also, the dynamics of observed TCP packets with and without block are shown as the red solid line and the blue dashed line, respectively. From Figure 3.9, we can see that the TCP traffic from the spam source is totally blocked in the suppression phase in both cases.

3.4 Potential Evasions

In such an ongoing arms race between spammers and anti-spammers, we envision that sufficiently aggressive spammers will seek sophisticated techniques to evade DBSpam. This is especially true for a spammer who is able to fully control remote spam proxy machines and deploy arbitrarily customized software. It may use non-off-the-shelf proxy programs, which can manipulate the traffic between the spam source and the first-hop proxy, to break packet symmetry. One possible way is to split a single reply packet from SMTP server into n fragmented packets on the first-hop proxy and then to transfer them back to the spam source.

However, as long as enough observations are collected, DBSpam can still capture such potential evasions. Recall that the effect of this packet splitting on SPRT model is just the change of the value of θ_0 , which measures the probability of 1 to n outbound TCP packets observed in a reply round. So, instead of $\theta_0 = \Pr(M = 1)$, now $\theta_0 = \Pr(M = 1) + \dots + \Pr(M = n)$. According to Equation (3.10), without changing other parameters, the augmented value of θ_0 renders more average number of observations needed to detect a spam proxy. On the other hand, not all SMTP

transactions have enough reply rounds for detection. Due to extended observations, short-living spamming activities may not be detected.

Table 3.6: False positive comparisons ($M = 5, K = 4, \Delta = 2s$)

θ_0	α^*	$E[N H_1]$	S-1-A	S-1-B	S-1-C	S-2-A	S-2-B	S-2-C	N-1	N-2
e^{-1}	0.005	5.5	0/110	0/103	0/123	0/27	0/186	0/94	0/0	0/0
0.5	0.005	8.1	0/0	0/103	0/120	0/0	0/97	0/32	0/0	8/8
0.5	0.01	7.1	0/110	0/103	0/121	0/21	2/159	0/89	0/0	12/12
0.5	0.02	6.0	0/110	2/105	0/121	0/27	7/194	1/94	0/0	21/21

To demonstrate the capability of DBSpam in capturing such evasions, we relax the definition of packet symmetry, in which one or two data packets may appear in one reply round, and adjust θ_0 to 0.5^{10} . Then, we estimate the overall false positives of DBSpam, which are listed in Table 3.6 under the parameter setting of $M = 5, K = 4$, and $\Delta = 2s$. For comparison, the results without relaxation are listed in the first row, while the results with relaxation are listed in the second row. Clearly, the short-living spamming activities are missed by DBSpam, with zero detection for S-*-A traces and much fewer detections for S-2-B and S-2-C traces. However, those spamming activities with more reply rounds can still be accurately detected. Since parameter α^* , the expected false positive probability, has the inverse effect on $E[N|H_1]$ according to Equation (3.10), we increase its value from 0.005 to 0.01 and 0.02, to accommodate short SMTP transactions for DBSpam detection. The third and fourth rows of Table 3.6 list the results after this adjustment, showing that DBSpam can capture short-living spamming activities by appropriately tuning α^* . When α^* is set to 0.02, DBSpam detects almost all spamming activities as before. In addition, those many more captures

¹⁰Note that θ_0 never exceeds 0.5 in all our traces with various packet lengths from 150 to 300 bytes.

are only at the cost of slightly more false positives, which is the necessary tradeoff in capturing evasive spam proxy traffic.

Moreover, instead of employing off-the-shelf proxy software, any advanced evasion technique will inevitably induce the modifications on the current spam methods and degrade the spam laundering efficiency. The customized proxy software also increases the cost of spamming. Overall, DBSpam indeed significantly raises the protection bar against e-mail spam, breaking the laundering and tracing out the real spam sources, in the anti-spam-vs-spam arms race.

3.5 Related Work

Many anti-spam techniques have been proposed and deployed to counter e-mail spam from different perspectives. Based on the placement of anti-spam mechanisms, these techniques can be divided into two categories: recipient-based and sender-based. In terms of fighting spam at the source, HoneySpam [16] might be the closest work to ours. In the following, we first briefly describe recipient-based and sender-based techniques, respectively, and then compare our work with HoneySpam.

3.5.1 Recipient-based Techniques

This class of techniques either (1) block/delay e-mail spam from reaching the recipient's mailbox or (2) remove/mark e-mail spam in the recipient's mailbox. Based on the classification of responses to spam given by [99], we further divide the receiver-based anti-spam techniques into pre-acceptance and post-acceptance subcategories. The pre-acceptance techniques mainly focus on blocking or delaying spam before the recipient's MTA accepts them in its mailbox, while post-acceptance attempts to weed spam out of received messages.

Pre-acceptance Techniques The pre-acceptance techniques usually utilize noncontent spam characteristics, such as source IP address, message sending rate, and violation of SMTP standards, to detect e-mail spam. Because these techniques are applied during SMTP transactions, they need to be deployed on the recipient's MTA.

DNSBLs: DNSBLs refer to DNS-based Blackhole Lists, which record IP addresses of spam sources and are accessed via DNS queries. When an SMTP connection is being established, the receiving MTA can verify the sending machine's IP address by querying its subscribed DNSBLs. Even DNSBLs have been widely used, their effectiveness [44, 72] and responsiveness [71] are still under study.

MARID: MARID (MTA Authorization Records In DNS) [58] is a class of techniques to counter forged e-mail addresses, which are commonly used in spam, by enforcing sender authentication. MARID is also based on DNS and can be regarded as a distributed whitelist of authorized MTAs. Multiple MARID drafts have been proposed, in which SPF [109], Sender ID [55] and DomainKeys [26] have been deployed in some places.

Tempfailing: Tempfailing [99] is based on the fact that legitimate SMTP servers have implemented the retry mechanism as required by SMTP, but a spammer seldom retries if sending fails. It usually works with a greylist that records the failed messages and the MTAs failed on their first tries.

Delaying: As a variation of rate limiting, delaying is triggered by an unusually high sending rate. Most delaying mechanisms, such as tarpitting [40], throttling [107, 110] and TCP Damping [51] are applied at receiving MTAs.

Sender Behavior Analysis: This technique distinguishes spam from normal e-mail by examining behavior of incoming SMTP connections. Messages from the machine exhibiting character-

istics of malicious behavior such as directory harvest are blocked before reaching mailbox [66].

Post-acceptance Techniques The post-acceptance techniques detect and filter spam by analyzing the content of the received messages, including both message header and message body. This kind of techniques can be deployed either at MUA (Mail User Agent) level in favor of individual preference or at MTA level for unified management.

E-mail address based filters: There are a variety of e-mail address based filters with different complexity. Among them, the traditional whitelists and blacklists are the simplest. Whitelists consist of all acceptable e-mail addresses and blacklists are the opposite. Blacklists can be easily broken when spammers forge new e-mail addresses, but using whitelists alone makes the world enclosed. [30] developed a new whitelisting system, which can automatically populate whitelists by exploiting friend-of-friend relationships among e-mail correspondents. [42] proposed a new spam filter based on Single-Purpose Address (SPA), which encodes a security policy that describes the acceptable use of the address. Any e-mail that violates the policy can be either marked, bounced, or discarded. [31] developed a remailer system, which maps a user's private permanent address to multiple public restrictive (e.g. duration) aliases for different correspondents and manages those aliases according to the user defined policy.

Challenge-Response (C-R): C-R [92] is used to keep the merit of whitelist without losing important messages. Incoming messages, whose sender e-mail addresses are not in the recipient's whitelist, are bounced back with a challenge that needs to be solved by a human being. After a proper response is received, the sender's address can be added into the whitelist.

Heuristic filters: The features that are rare in normal messages but appear frequently in spam, such as nonexisting domain names and spam-related keywords, can be used to distinguish spam from normal e-mail. [90] is such an example. Each received message is verified against the heuris-

tic filtering rules. Compared with a predefined threshold, the verification result decides whether the message is spam or not.

Machine learning based filters: Since spam detection can be converted into the problem of text classification, many content-based filters utilize machine-learning algorithms for filtering spam. Among them, Bayesian-based approaches [19,35,52,112] have achieved outstanding accuracy and have been widely used. [37] studied the effect of combining multiple machine learning models on reducing false positives of spam detection. As these filters can adapt their classification engines with the change of message content, they outperform heuristic filters.

Signature-based filters: Similar to the concept of a virus signature, a spam signature is the identity of a spam message and is usually derived from certain computation on the spam message. For each incoming message, a signature-based filter first derives its signature, then queries the registered server for signature test, and takes proper actions based on the response. To be effective, signature-based filters usually collaborate and contribute signatures through peer-to-peer networks [67,74,114].

3.5.2 Sender-based Techniques

Usage Regulation: To effectively throttle spam at the source, ISPs and ESPs (E-mail Service Providers) have taken various measures such as blocking port 25, SMTP authentication, to regulate the usage of e-mail services. Message submission protocol [32] has been proposed to replace SMTP, when a message is submitted from an MUA to its MTA.

Cost-based approaches: Borrowing the idea of postage from regular mail systems, many cost-based anti-spam proposals [18,48,60,103] attempt to shift the cost of thwarting spam from the receiver side to the sender side. All these techniques assume that the average e-mail cost for a

normal user is negligible, but the accumulative charge for a spammer will be high enough to drive him out of business. Cost concept may have different forms in different proposals. SHRED [48] proposes to affix each mail with an electronic stamp and punish spammers by reducing their stamp quotas and charging them real money, while Penny Black Project [60] enforces a sender to pay e-mail postage by associating a CPU or memory intensive computation with an e-mail sending process. The computation result, called “Proof-of-work”, is attached with the message and can be easily validated by the recipient.

3.5.3 HoneySpam

HoneySpam [16] is a specialized honeypot framework based on honeyd [68] to deter e-mail address harvesters, poison spam address databases, and intercept or block spam traffic that goes through the open relay/proxy decoys set by HoneySpam. With the network virtualization offered by honeyd, HoneySpam can set up multiple fake Web servers, open proxies, and open relays. Fake Web servers provide specially crafted Web pages to trap e-mail address harvesting bots. Fake open proxies or open relays are used to track spammers exploiting them and block spam going through them.

HoneySpam shares the same motivation of countering spam at the source as DBSpam, and both deal with spam proxies. However, the role of proxy and anti-spam approaches in HoneySpam are quite different from those in DBSpam. The proxies of HoneySpam are intentionally set on end hosts, and spam sources are logged by HoneySpam. Thus, spam tracking is very easy. In contrast, detecting spam proxies is the major task of DBSpam, and proxy identification and spam tracking can only be accomplished through traffic analysis. On the other hand, these two tracing and blocking systems are complementary to each other. Moreover, both of them can be used for spam

signature generation, spam forensic and law enforcement.

3.6 Summary

In this chapter, we presented a simple yet effective system, DBSpam, to detect and break proxy-based e-mail spam laundering activities inside a customer network and to trace out the corresponding spam sources outside the network. Instead of content checking, DBSpam leverages the protocol semantics and timing causality of proxy-based spamming to identify spam proxies and real spam sources behind them. Based on connection correlation and packet symmetry principles, DBSpam monitors the bidirectional traffic passing through a network gateway, and utilizes a simple statistical method, *Sequential Probability Ratio Test*, to quickly filter out innocent connections and identify the spam laundry path with high probability. To further reduce false positives and false negatives, we propose a noise reduction technique to make spammer-tracking more accurate after gathering consecutive correlation detection results. We implement a prototype of DBSpam using *libpcap* on Linux, and conduct trace-based experiments to evaluate its effectiveness. Our experimental results reveal that DBSpam can be tuned to detect spam proxies and sources with low false positives and false negatives in seconds. After detecting spam proxies and related spam sources, DBSpam can effectively throttle or block spam traffic.

Chapter 4

Countering Malicious Instant Messages

Instant messaging (IM) has been one of most frequently used malware attack vectors due to its popularity. IM malware usually finds and hits next victim by exploiting current victim's contact list and playing social engineering tricks. It is very difficult to detect and suppress the spread of IM malware through conventional approaches such as blocking and filtering, because sources of the malicious IM messages are legitimate and their contents are in disguise. IM systems including public IM systems (e.g., AOL Instant Messenger) and enterprise IM systems (e.g., Reuters Messaging) are widely used by enterprises and organizations for internal communication. However, previously proposed protection approaches [54, 57, 100, 108] are ineffective to defend against IM malware in an enterprise-like network environment, mainly because of high false positive rate and the requirement of the IM server being inside the protected network.

In this chapter, we present HoneyIM, a framework for automating the process of IM malware detection and suppression in an enterprise-like network. HoneyIM is based on the concept of honeypot and detects IM malware by leveraging its inherent spreading characteristics. Specifically, HoneyIM uses decoy accounts in normal users' contact lists as sensors to capture malicious

content sent by IM malware, which achieves almost zero false positive. With accurate detection, HoneyIM suppresses malware by performing network-wide blocking. In addition, HoneyIM delivers attack information to network administrators for system quarantine and recovery. The core design of HoneyIM is generic and can be applied to a network that uses either private (enterprise) or public IM services.

This chapter is structured as follows. We first overview the related work in securing instant messaging systems in Section 4.1. Then, we present the framework of HoneyIM in Section 4.2. Next, we detail the implementation and evaluation of HoneyIM in Sections 4.2.5 and 4.3, respectively. We further discuss possible evasion to HoneyIM and the countermeasures in Section 4.4. Finally, we summarize the contributions of this chapter in Section 4.5.

4.1 Related Work

The security threats posed by IM malware have been studied in [39, 56]. In [39], the spreading speed of IM malware is estimated, showing that 500,000 machines could be infected within a minute.

Previous defense schemes against IM malware are closely related to IM network modeling and traffic measurement. Based on individual measurement and analysis, [61, 83, 108] all verify that IM social networks formed by IM contacts are scale-free, that is, the IM network connectivities follow power-law distributions. However, a recent measurement study [111] suggests that Weibull distributions may be more appropriate for describing the connectivity of IM social networks. For scale-free networks, a small portion of nodes that are highly connected have significant effect on mitigating malware spread. Based on this observation, Smith [83] proposed to delay the propagation of IM malware by disabling the accounts of most connected IM users on the network. This

scheme needs to be deployed on IM servers. It only reduces the spread speed and may have significant side-effects. Williamson *et al.* [108] applied their virus throttling mechanism to IM and demonstrated its effectiveness by simulation. The throttling to IM is also conducted at servers. The throttling becomes blind blocking if its threshold is very restrictive, which degrades the usability. Mannan and van Oorschot [57] proposed two defense methods, namely limited throttling and CAPTCHA-based challenge-response. They also provided a usage study on per-user frequency of IM text messages and file transfers to support the applicability of their second scheme. Liu *et al.* modeled the spread of IM malware using multicast tree [54] and analogous branching process with varied lifetime [53]. HoneyIM is orthogonal to all the schemes mentioned above, and can achieve accurate detection and blocking without degrading usability.

Trivedi *et al.* studied the network and content characteristics of spim, the spam messages on IM networks, by using a proxy server as honeypot [96]. Their work is different from HoneyIM, since [96] is a measurement study and it targets spim but not IM malware. The honeypot used in [96] refers to a SOCKS proxy, which is exploited by spimmers to conceal their identities.

4.2 HoneyIM Framework

HoneyIM aims to assist network administrators in IM malware defense by automating the process of malware detection and suppression in an enterprise-like network. Utilizing the innate spreading characteristics of IM malware and applying the concept of honeypot, HoneyIM can detect and block unknown IM malware at its early stage of spreading, which greatly facilitates network filtration and system quarantine and recovery. In this section, we first give an overview of HoneyIM, how and why it can detect IM malware early. Then, we discuss several issues that need to be considered when using HoneyIM in practice. After that, we present the design of HoneyIM

and the functionalities of its components. Finally, we describe the deployment of HoneyIM in an enterprise-like network.

4.2.1 Overview

HoneyIM is based on the concept of honeypot. As an effective intrusion detection technology, honeypot has been used widely. According to [95], *a honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource*. Not only can a honeypot be a physical machine or a specialized program, which is the common case, but it can also be an e-mail address, or even an IM decoy user. Since IM malware always attempts to infect other users on the victim's contact list, HoneyIM exploits decoy users to detect IM malware. Under normal circumstances, a client user will not initiate a conversation with a decoy user. Therefore, if the decoy user receives a file transfer request or a URL-embedded text message originated from a client user, it is highly probable that malware is spreading and the request/message sender is compromised. Thanks to decoy users, HoneyIM can achieve almost zero false positive in detection. This strong guarantee, which is rarely offered by other schemes, relieves network administrators from worrying about possible interruption to normal IM users caused by the protection technique. In addition, HoneyIM can block malicious content that has been detected and inform network administrators of the attack information, e.g., the IP address of the compromised machine, in real-time.

Figure 4.1 illustrates the working mechanism of HoneyIM. The IM user with an icon of honeypot is the one whose contact list contains a decoy user. The events happen in the following sequence. (1) Some IM malware compromises an IM client and (2) propagates. However, (3) when it tries to spread again, it hits a decoy user and (4) is detected by HoneyIM. (5) HoneyIM blocks the malicious content in IM traffic (either at the edge gateway or at the IM server if the

IM service is provided within the network) and non-IM traffic¹ instantly, and notifies the attack information to the network administrator.

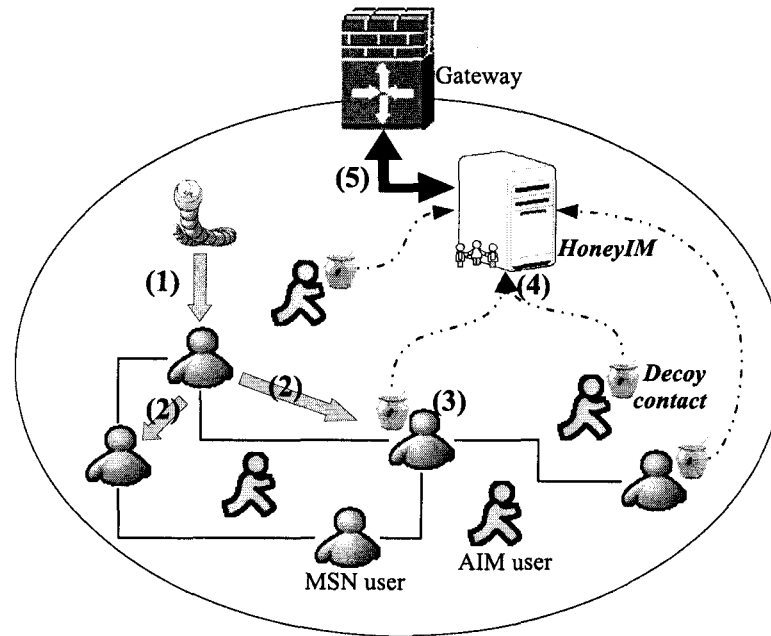


Figure 4.1: Working mechanism of HoneyIM

HoneyIM is designed to be independent, with no restriction on the type and location of IM servers. Therefore, the framework of HoneyIM can be flexibly realized under the context of either public IM services or private (enterprise) IM services being used in the protected network. The core of HoneyIM is the same for either server-enhanced (with private servers) or serverless (with public servers) realization. The difference lies in the implementation and deployment, which will be discussed in Section 4.2.4. The framework of HoneyIM consists of several modules and these modules can be deployed in a single machine or at different places.

¹Doing this is to block accesses to malicious contents, e.g., malicious URLs.

4.2.2 Design Issues

The success of HoneyIM largely depends on the use of decoy users. In the following, we discuss three issues of HoneyIM that are much related to decoy user, including initialization, sensitivity, and compatibility.

The initialization of HoneyIM mainly refers to the creation and addition of decoy user accounts. Strictly speaking, it is a deployment issue. If public IM services are used in the protected network, the network administrators need to create decoy accounts and solicit some volunteer IM users to add those decoy users into their contact lists. In contrast, if an enterprise IM service is employed, the creation and addition of decoy users can be done automatically by the IM server. However, the system must notify volunteer users the purpose and usage of decoy accounts, and provide a disable (or opt-out) option. This HoneyIM initialization is fulfilled at one time, and the update of decoy accounts could be performed if necessary. In addition to the volunteer policy for IM user cooperation, the network administrators might require the IM users who have high connectivity degrees (i.e., the super-nodes in IM networks) to include decoy accounts in their contact lists.

The sensitivity of HoneyIM is measured by the ratio between the number of infected users and that of all IM users in the protected network when the spreading of IM malware is first detected. The key factor affecting the sensitivity of HoneyIM is the coverage of HoneyIM—the portion of the IM users equipped with decoy user accounts among all IM users within the network. It is obvious that HoneyIM cannot detect malware for those users who do not include decoy accounts in their contact lists. Moreover, IM malware may intentionally or inadvertently bypass HoneyIM by not hitting decoy users in the infected users' contact lists. The word “intentionally” does not mean that the IM malware knows the decoys in advance, but reflects its capability of distinguishing

decoys from other contacts. Here we assume that the threat comes from the outside of the protected network and the inside IM users do not collude with the outside attackers. Given the coverage of HoneyIM, which is usually determined by the network administration policy, we will consider how to counter evasive IM malware to improve HoneyIM sensitivity in Section 4.4.

Compatibility is not an issue if HoneyIM is deployed on an enterprise IM server, since the server can maintain the compatibility with supported IM clients. However, the compatibility has to be taken into account if public IM services are used in the protected network. Under this circumstance, various types of public IM systems may coexist. This is especially true on the networks with less strict IM usage policies such as campus networks. Thus, HoneyIM should be able to talk with different types of IM clients.

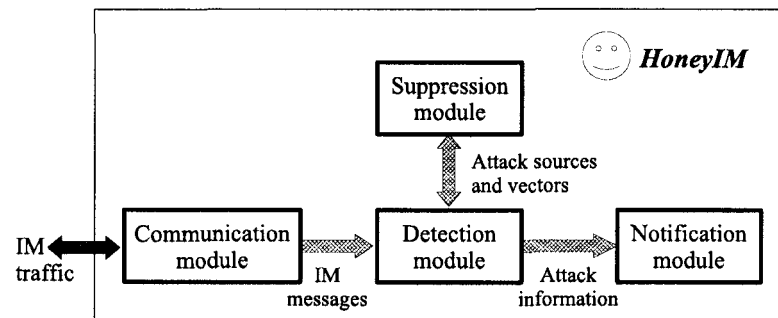


Figure 4.2: Framework of HoneyIM

4.2.3 System Components

Figure 4.2 shows the general framework of HoneyIM, which comprises four modules each performing a specific functionality. Note that these modules could be deployed either on the same machine or on different hosts (or network devices). As displayed, the communication module is responsible for handling IM traffic. It parses the IM traffic to decoy users and delivers it to

the detection module. The detection module extracts attack vectors and related information from IM messages, and then feeds them into the suppression and notification modules. The suppression module sifts through network traffic and filters out malicious traffic containing attack vectors. Meanwhile, the notification module informs network administrators of the detected malware spreading.

Communication Module

The communication module is the base of HoneyIM. Decoy accounts use it to join IM networks and communicate with normal IM clients. This module realizes all necessary functions of a normal IM client, such as signing on/off, setting presence status, receiving messages and files, etc. These functions are automatically executed by default and can also be manually operated by a network administrator. The module only accepts the messages from the users on the contact list for blocking “spim”, the spam on IM networks. The communication module should support all IM protocols that are used by the protected IM services, and allow multiple accounts to log into different IM networks simultaneously if necessary.

Detection Module

The detection module serves three purposes: (1) detecting compromised IM clients, (2) identifying attack vectors, and (3) validating attack vectors. It accomplishes the first two tasks by consulting the communication and suppression modules and scrutinizing IM messages delivered by the communication module, and attains the last task by conducting deep-inspection.

The detection module classifies a sending IM client as compromised, when a decoy account receives a file transfer request or a text message with URL from the IM client. The reason is that it is very rare for a normal user to issue such a request or message to the decoy account². The

²Even if a normal user accidentally sends a message to the decoy account, the message is usually a pure text

detection is not affected by client-to-client or client-to-server traffic encryption because the IM messages received by a decoy (as a client) must be in plain-text. If IM malware spreads through file transfer, the attack source, i.e., the IP address of the compromised machine, is immediately known as a file transfer is usually done between two IM clients directly. However, if IM malware spreads through URL message, we cannot identify the sender directly because the message is usually relayed through server. Under this circumstance, the attack source is inferred with the help of the suppression module, which will be described shortly. The detection module can easily generate the attack vector information such as malicious file names and malicious URLs from the received IM messages.

Furthermore, the detection module performs deep-inspection to verify the virulence of the received file or URL. There are many techniques available to achieve this purpose. For example, we can use dynamic taint analysis based techniques such as TaintCheck [63] and Argos [65] to examine if a received binary can compromise system and to generate the corresponding signature if a compromise occurs. We also can adopt the technique used by HoneyMonkey [104] to check received URLs. HoneyMonkey detects Web exploits by browsing URLs inside a virtual machine and monitoring the change of system states. In general, any effective and efficient host-based anomaly detection techniques can be used for deep-inspection. HoneyIM does not contain any specific technique for analyzing IM malware, but rather provides a platform to apply existing techniques for malware dissection and leave the choice of what technique to use to network administrators. The adopted techniques are implemented as plug-ins of the detection module, and the deep-inspection is conducted in a contained environment such as a virtual machine to prevent HoneyIM itself from being compromised.

message.

The incorporation of deep-inspection is justified by the following considerations. First, deep-inspection can further reduce false positives. It is possible that innocent URLs or files could be sent with malicious content by IM malware to disguise their malice. Second, deep-inspection helps discover additional or real attack vectors used by IM malware. For example, file deep-inspection can generate the signature of malware binary, based on which the filtering is much more robust against evasion than based on file name. IM malware can also use different URLs in its spreading, which in fact are doorway Web pages redirecting traffic to the same Web site that hosts real exploits. With URL deep-inspection, the protection can be further enhanced because not only doorway URLs but also real exploit URLs can be discovered. Last but not least, deep-inspection uncovers the IM malware activities, such as the infection mechanism and the infected files, for network administrators.

After attack vector extraction and validation, the detection module supplies the validated attack vectors and sources to the suppression module for immediate network traffic filtration. In the meantime, the detection module feeds all collected attack information into the notification module, which informs network administrators of the occurrence of an attack in real-time for prompt system quarantine and recovery.

Suppression Module

The suppression module in essence is a network filter. It takes the attack source and vector information from the detection module as input. Then, it blocks any traffic from attack sources and filters out network traffic that contains attack vectors. Different from other modules that have no requirement for deployment location, the suppression module should be installed at a network vantage point, where it can monitor all traffic passing through the protected network. The location of the suppression module will be further discussed in Section 4.2.4.

The suppression module consists of two components: non-IM traffic filter and IM traffic filter. These two components are logically independent for flexible implementation and deployment. The non-IM traffic filter fulfills two tasks: blocking attack sources and filtering non-IM network traffic. For the former, the filter simply drops any packet from the attack sources to terminate malware propagation. For the latter, the filter examines contents of inbound and outbound packets to identify if an internal user is attempting to access a malicious Web page or transfer a virulent file. Any packet containing a matched attack vector will be discarded.

The IM traffic filter also provides two functionalities. The first is traffic filtration, which weeds out the IM messages that either come from (or go to) the compromised clients or contain identified malicious file names or URLs. Although a file is usually transferred between two clients, the IM messages for establishing transfer connections are relayed through servers in plain-text for mainstream IM products. Therefore, blocking malicious file transfer by dropping connection establishment messages is not affected by client-to-client encryption. The second functionality of the IM traffic filter is to help identify malicious URL sending hosts within the protected network. Because messages are relayed through server, the detection module cannot identify the sources of malicious URL messages. To track the IP address of the compromised host, the IM traffic filter records the URLs and the corresponding IP addresses of their senders. With this information, the detection module can easily pinpoint the malicious URL senders.

Notification Module

The notification module plays the role of messenger. Its job is to inform network administrators of the occurrence of IM malware spread upon the detection of an attack. Given the fast spread of IM malware, the notification to network administrators should be made in real-time or near real-time by means of SMS (Short Messaging Service) or IM. The notification module can

also notify the victim about the fact that his machine has been infected with IM malware via IM or e-mail.

4.2.4 Deployment

As mentioned in the overview section, HoneyIM can be deployed with a private IM server inside the protected network (server-enhanced deployment) or with public IM services outside the network (serverless deployment). The major differences between the two deployments lie in the function location and system initialization of HoneyIM. In serverless deployment, the non-IM and IM traffic filters of the suppression module have to be placed on the network edge device. However, in server-enhanced deployment, while the non-IM traffic filter still needs to be on the network edge device, the best place for the IM traffic filter is the private IM server, where the filter can see all IM traffic. Moreover, in practice many IM servers already include the message filtering functionality, making IM traffic filtering much easier there.

The deployment of HoneyIM also involves system initialization, i.e., the creation and addition of decoy accounts. In serverless deployment, network administrators need to register accounts for decoy users on public IM services before running HoneyIM. Due to the maximum size of contact list (e.g., 600 for MSN) and the protection consideration, the administrators can create multiple decoy accounts and use them for different groups of IM users. Then, the decoy accounts are added into the volunteer IM users' contact lists with their cooperation. By contrast, the server-enhanced deployment saves the efforts of network administrators and IM users by automating the creation and addition of decoy accounts, just like the use of AIM Bots for shopping and movie guide. This can be achieved by adding a decoy account management module to the private IM server. The module can also be used to (1) provide IM users with the information of decoy accounts and

the option to enable/disable them, and (2) update decoy accounts periodically against potential evasion.

4.2.5 Prototype

To demonstrate the efficacy of HoneyIM, we have built a prototype of the serverless HoneyIM, which can be easily transformed to the server-enhanced HoneyIM prototype with minor changes in function location and system initialization. We implement the HoneyIM modules using different techniques. We use a full-fledged open-source IM client Pidgin (formerly known as Gaim) [8] to build the communication module. The detection module employs Capture [93], a high interaction client honeypot on Windows systems, for URL deep-inspection. The detection module extracts URLs from the communication module and feeds them into Capture, which decides whether a URL is malicious by comparing the system states such as registry and running processes before and after the URL is accessed. For any file transfer request HoneyIM does not perform deep-inspection but immediately fires an alert instead, given that the file transfer method is relatively unpopular in IM malware spreading and most IM users and programs are vigilant to this type of threat. HoneyIM receives the delivered file and sends it to network administrators via e-mail. In the construction of the suppression module, we use Perl IPQueue module for iptables [62] to perform URL logging and pattern-matching. We implement the notification module with two communication means: e-mail and SMS. The suppression module communicates with the detection module via network socket, and thus can be deployed on a separate machine.

Because Pidgin supports multi-protocol and multi-account, HoneyIM can log into multiple accounts on multiple IM networks simultaneously. Therefore, it can provide protection for multiple public IM networks. Note that the choice of Pidgin and Capture is mainly due to the

availability of their source code. Upon the accessibility of source code, any IM clients or anomaly detection systems can be used to construct HoneyIM.

4.3 Evaluation

In this section, we first evaluate the detection sensitivity of HoneyIM under different coverages via simulation. Then, we validate the applicability of HoneyIM through real experiments.

4.3.1 Simulation

When adding decoy accounts is voluntary for IM users on the protected network, it is very possible that HoneyIM does not cover all IM users. Under this circumstance, how effective would HoneyIM be? Because we cannot carry out a large-scale experiment in practice, we turn to simulation for answering this question. We adopt the simulation model from [115] due to the similarity in propagation between IM malware and e-mail worms [115]. The major metric we use is the percentage of IM users being infected by the time the IM malware is firstly detected by HoneyIM (the percentage of infected IM users for short), and we investigate its variation under different HoneyIM coverages.

1) Simulation Model

The simulation model of IM malware propagation is described as follows. First, when an IM user receives an IM message, she may or may not read the message immediately. The reading delay for user i , denoted by T_i , is a stochastic variable. When the user receives a message with a malicious URL ³, she clicks the URL with a clicking probability denoted as C_i . We assume that C_i is a constant for user i . If the malicious URL is clicked, the malicious code is downloaded and

³The situation for malicious file transfer is similar.

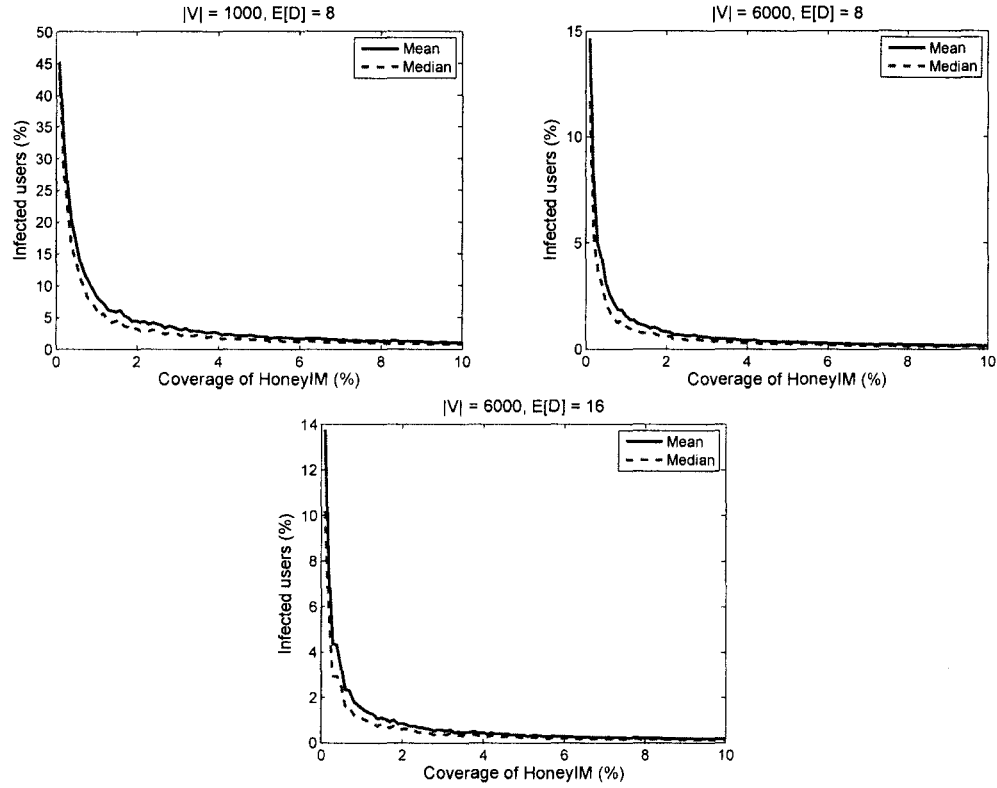


Figure 4.3: Relations between HoneyIM coverages and infected user percentages

executed immediately. It infects the current IM client and sends malicious URLs to all the victim's contacts with no delay. The malware will not spread again unless the user receives the same URL and clicks it again.

Before we start the simulation, we need to determine the IM network topology and the values of each C_i and T_i . Here the IM network refers to the virtual network composed by the contact lists of the IM users on the protected network. According to [83] that studies an IM network containing 50,158 users, over 80% of the user contacts are bidirectional, indicating that most of users are also in the contact lists of their buddies. Thus, we model the IM network topology by an undirected graph $G = \langle V, E \rangle$. For $\forall v \in V$, v denotes a node (IM user), and for $\forall e = (u, v) \in E, u, v \in V$, e represents an edge that connects two users, u and v , who are in each other's contact list. $|V|$ is

the total number of nodes, and $D(i)$ is the degree of node i , i.e., the number of edges connected to node i . The size distribution of contact lists has been identified as scale-free by [61, 83, 108], except that [111] claims that Weibull distribution has a better fit. However, [111] does not give the parameters of Weibull distribution and the number of their monitored IM users is small compared to [61, 83, 108]. Therefore, we model the IM network topology as power law and set the power law exponent α to 1.7, based on the measurement results from [61] and [83]. The network is generated by using GLP power law generator [22] with the given α , the number of nodes $|V|$, and the average node degree $E[D]$. We generate three IM networks with the number of nodes $|V| = 1000/6000/6000$ and the average node degree $E[D] = 8/8/16$, respectively. The maximum node degrees of the generated networks are all below 600, the maximum size of a contact list for MSN.

Similar to [115], we assume that IM users have independent behaviors. Due to the large number of users $|V|$ and independent behaviors, the mean values of user reading delay T_i and clicking probability C_i , denoted by $E[T_i]$ and $E[C_i]$ ($i = 1, 2, \dots, |V|$), can be assumed to follow Gaussian distribution. That is, $E[T_i] \sim N(\mu_T, \sigma_T^2)$ and $E[C_i] \sim N(\mu_C, \sigma_C^2)$. We also assume that T_i follows exponential distribution and C_i is a constant for user i , and the generation of T_i and C_i is constrained by $T_i \geq 0$ and $C_i \in [0, 1]$. In simulation, we use $N(20, 10^2)$ and $N(0.5, 0.3^2)$ to generate $E[T_i]$ and $E[C_i]$, respectively.

2) Simulation Results

Given the network topology, we *randomly* deploy decoys in the network with different coverage \mathfrak{R} and run simulation experiments. Each simulation run stops once IM malware hits a decoy user (blocking is in effect immediately) or timeout occurs. The number of infected users and detection time are the simulation output. For each coverage \mathfrak{R} , we vary the decoy deployment 10

times and run simulation 100 times for each deployment, and have the mean and median values derived from these 1,000 simulation experiments.

With the increase of HoneyIM coverage, the corresponding percentages of infected IM users on three different IM networks are shown in Figure 4.3, in which the solid curves are for mean values and the dashed curves are for median values. The mean curves are above the median curves for very small coverage values, and both types of curves drop sharply and converge to zero with the increase of coverage. This clearly demonstrates the effectiveness of HoneyIM. Figure 4.4 further zooms in on y-axis and compares the mean curves of the three IM networks. Even with the 5% coverage, HoneyIM can detect the spread of IM malware only after 2% (or 0.4%) of all IM users are infected for the network with $|V| = 1,000$ (or $|V| = 6,000$). Compared to the number of nodes $|V|$, the average node degree $E[D]$ has much less effect on the performance of HoneyIM. Two mean curves, the dashed one for $|V| = 6,000, E[D] = 8$ and the dotted one for $|V| = 6,000, E[D] = 16$, are almost identical.

We also compare the performance of HoneyIM with that of IM throttling [108]. The throttling of IM malware is usually conducted on an IM server. We use the “no-delay” mode of IM throttling and configure the working set size and threshold to 5 and 2, respectively, as suggested. Since it is difficult to simulate the working set for each user at run time, we simplify the propagation model by (1) randomly determining a node’s working set between 0 and 5 right before the node is propagating and (2) blocking the node after its propagation (no matter whether the delay queue is overflowed or not). Therefore, the maximum number of the nodes that a compromised node could infect is its working set size plus 2 (the threshold). Note that this model is conservative compared to the original scheme, as we block an infected node permanently once it starts spreading.

Figure 4.5 shows the performance comparisons between HoneyIM (coverage $\mathfrak{R} = 3\%$) and

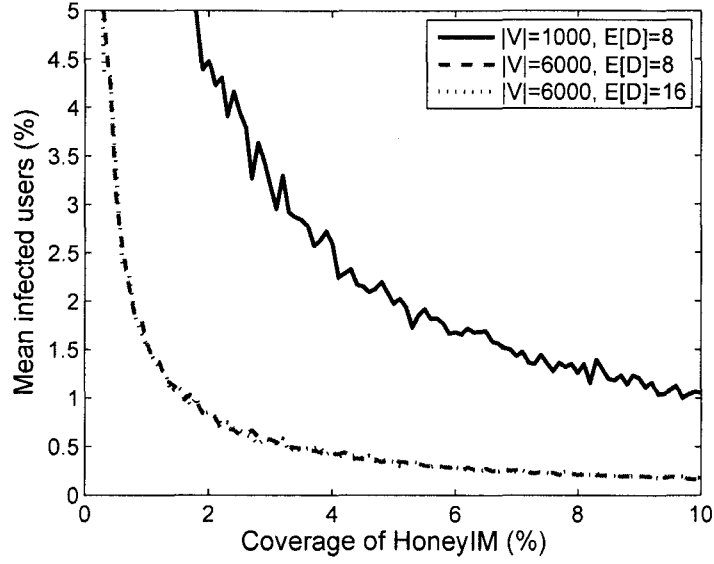


Figure 4.4: Comparisons among mean curves

throttling on the three IM networks. The solid curves represent HoneyIM and the dotted curves represent throttling. The dashed curves show the spreading of IM malware with no mitigation. Note that the y-axis is logarithmic, and all the results for throttling and no mitigation are the mean values for 100 runs. Compared with throttling, HoneyIM can achieve similar performance in terms of the number of infected users on a small network ($|V| = 1,000$), and perform much better when the network becomes bigger ($|V| = 6,000$) and has more edges ($E[D] = 16$). More importantly, HoneyIM can accurately detect the malware and block its spread right after detection, while throttling cannot differentiate malicious traffic from normal traffic, let alone block them in an effective manner.

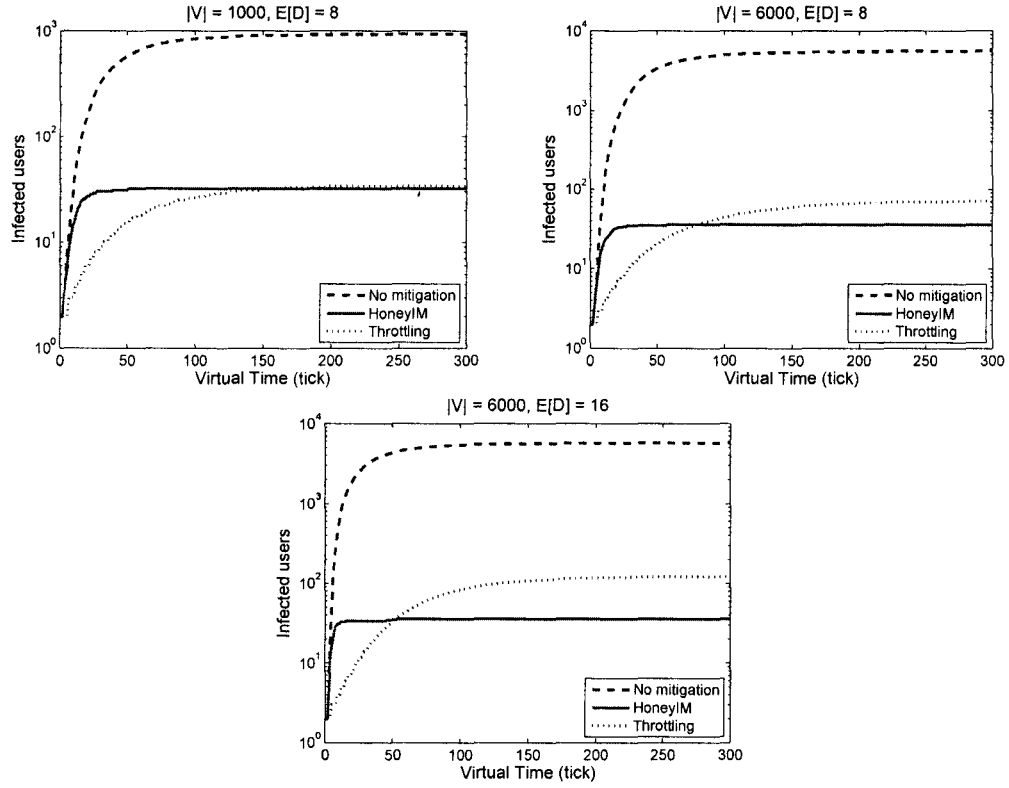


Figure 4.5: Effect comparisons between HoneyIM and IM throttling

4.3.2 Real Experiment

We set up a small testbed comprising three machines. We use one machine as the IM client and the other two as HoneyIM and the network gateway. The suppression module of HoneyIM is deployed on the network gateway. Both the IM client and HoneyIM run inside virtual machines for security and ease of experimentation. We first use real IM malware binaries we have collected to test HoneyIM by running malware on the IM client machine. We test Jitux-A [85], Kelvir-F [86], Kelvir-M [84], and Kelvir-Q [87], respectively, all of which spread through malicious URL messages on MSN platforms. The URLs for Jitux-A and Kelvir-F lead to .exe and .scr file downloading, while the URLs for Kelvir-M and Kelvir-Q point to .php scripts which also harvest

victim's e-mail addresses. Unfortunately, due to the legal reaction taken by the IM providers and security community, the Web pages pointed by these known malicious URLs are either invalid or have been removed by the hosting Web sites⁴. The URL message sent by Kelvir-F is not even received by HoneyIM, because of the filtering in MSN servers. No detailed information about IM malware is given by deep-inspection. Thus, we reconfigure the detection module to skip the deep-inspection step and rerun the tests. The suppression and notification modules work well as expected.

We also test the prototype using a generic approach which overcomes the difficulty caused by the invalidity of the known malicious URLs. We mimic IM malware by sending malicious URLs collected by ourselves to decoy accounts. The malicious URLs we used, in principle, have no difference from those carried by known IM malware in terms of Web exploits. Thus, they should have the same effect on normal IM clients and HoneyIM. The URL process time of HoneyIM is mainly determined by deep-inspection, which is usually finished within 30 seconds. Overall, HoneyIM successfully detects all malicious URLs, updates the URL blacklist, and sends the attack information to the designated recipient via SMS and e-mail. For emulated malicious file transfers, HoneyIM automatically receives files, reveals file names to the suppression module, and sends file payloads to the designated recipient via e-mail. The whole process takes seconds to complete, since no deep-inspection is performed for file transfer.

4.4 Discussion

In previous sections, we assume that IM malware always attempts to infect all online contacts by either initiating a file transfer or sending a malicious URL during its spread. This hit-all propa-

⁴This situation also applies to other known IM malware.

gation strategy, however, might not always be used. For example, “smart” IM malware may send malicious URLs or files only to the active online contacts, i.e., those contacts that the infected IM client is talking to; or the propagation is activated only after the infected client receives a message. Taking the non-hit-all strategy, IM malware might not hit the decoy contact even if the contact list of the infected IM user includes the decoy accounts.

IM malware can realize the non-hit-all propagation strategy by either intentionally or randomly selecting a part of all online contacts as targets. To prevent decoys from being easily distinguished, we can enhance HoneyIM with interaction functionality. As a countermeasure, HoneyIM uses the interaction functionality to mimic human users for decoys by initiating chat sessions with normal users, making it much harder for IM malware to tell decoys from others. The chat content can be important security notices or other user interested information. We readily agree that IM malware can still avoid decoy contacts even with the interaction functionality, for example, by infecting the most active contacts. However, the spread of this type of IM malware could be significantly reduced. According to a recent IM traffic measurement [111], IM users only contact a small portion of users in their contact lists. On average an AIM user chats with only 1.9 users and an MSN user chats with 5.5 users.

The random selection of infection targets may also help IM malware bypass decoy contacts. To study the effect of the random selection on HoneyIM, we conduct the following experiments based on the previous simulation for HoneyIM. We apply a probabilistic propagation strategy to the experiments. That is, when IM malware propagates, it will send malicious content to each contact with a probability p . With the probabilistic infection, the number of users that malware will contact becomes $p \times n$ on average, where n is the total number of the online contacts of the infected user.

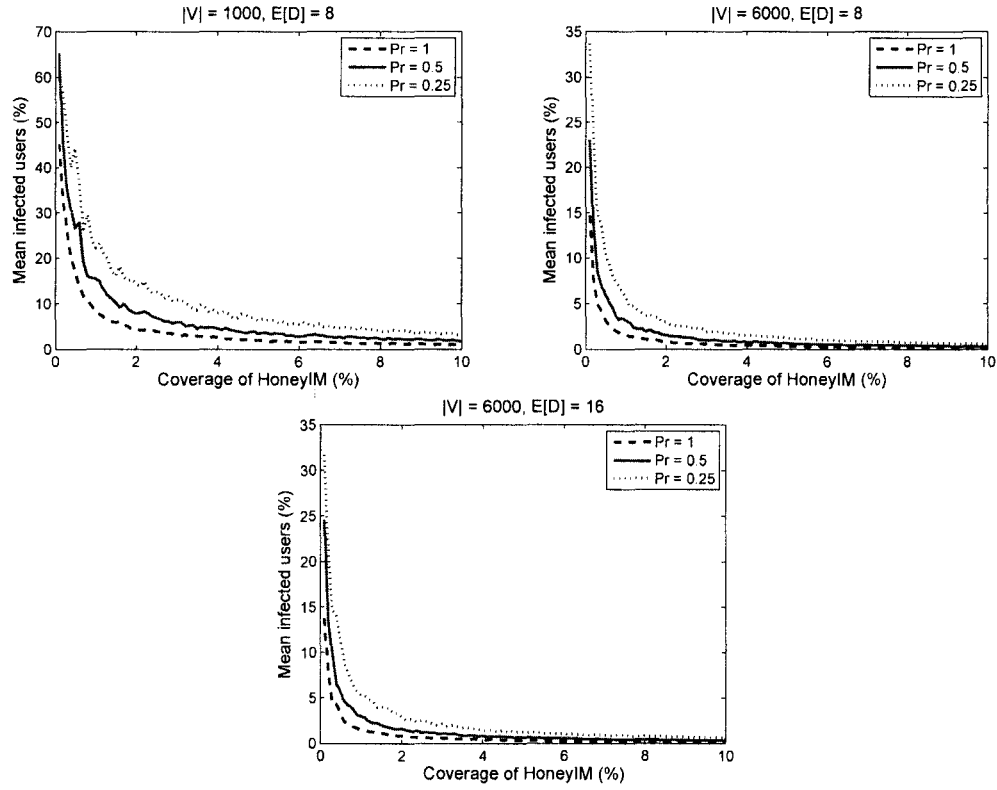


Figure 4.6: Effects of randomly selecting infection targets on HoneyIM

We test and compare the effects of random target selection on HoneyIM with three different probabilities $p = 1, 0.5, 0.25$ on the three IM networks, respectively. Here $p = 1$ refers to the aforementioned deterministic infection. The comparison is displayed in Figure 4.6, in which the curve of $p = 0.5$ is above the curve of $p = 1$ but below the curve of $p = 0.25$. It indicates that with the decrease of the probability value, the average number of infected users becomes larger. However, the difference among three curves quickly becomes negligible with the increase of the coverage. In general, the random target selection has little effect on HoneyIM.

4.5 Summary

In this chapter we presented HoneyIM, a novel detection and suppression mechanism to defend against IM malware for enterprise-like networks. Distinct from all previous defense schemes, HoneyIM introduces decoy users for IM malware detection. It exploits the basic spreading characteristics of IM malware and guarantees almost zero false positive. With accurate detection, the suppression of HoneyIM achieves instant network-wide blocking. Moreover, HoneyIM notifies network administrators of the infected machines and the infection features of IM malware in real-time. The generic design of HoneyIM enables its flexible realization on a network that uses either enterprise IM services or public IM services. We have built a prototype of HoneyIM that works with public IM services using open-source IM client Pidgin and client honeypot Capture. The simulation studies demonstrate that even with a small portion of IM users equipped with decoy accounts, HoneyIM can still detect and block IM malware in the early stage of its spread. The real experiments on the prototype further demonstrate that HoneyIM is competently capable of detecting and suppressing the spread of IM malware.

Chapter 5

Improving E-mail Reliability

In this chapter we present a Collaboration-based Autonomous e-mail Reputation system (CARE) that aims to significantly improve e-mail reliability. CARE works at domain level and rates both spam domains and nonspam domains. CARE enables a domain to build its reputation database, including both frequently contacted and unacquainted e-mail sending domains, by (1) locally recording e-mail sending behavior of remote domains and (2) exchanging the local information with other collaborating domains. CARE examines the trustworthiness of e-mail histories obtained from collaborators by correlating them with local e-mail history, and integrates the local and remote information to derive the reputation of remote domains.

This chapter is organized as follows. Section 5.1 surveys the related work. Section 5.2 presents the original motivation of this work. Section 5.3 details the design of CARE. Section 5.4 validates the effectiveness of CARE. Finally, Section 5.5 summarizes this chapter.

5.1 Related Work

E-mail reputation systems rate e-mail sending entities based on the history of their sending behaviors. The entity can be e-mail address, IP address, or domain name. Some reputation systems use qualitative measures (e.g., good or bad) while others use quantitative measures (e.g., spam score is 58). Both automatic and manual operations are used in reputation establishment and maintenance. A brief taxonomy of e-mail reputation systems is given in [15].

Vipul's razor (later branded as Cloudmark) [67] uses e-mail content as the rating identity. It maintains a collaborative network through which the signatures of human-identified spam are submitted and distributed. Cloudmark employs a trust evaluation system (TeS) to maintain the reputation of each signature contributor and evaluate the trustiness of new signatures. TeS requires a certain number of pre-selected trusted contributors for bootstrap and may flag signatures as contested if consensus cannot be reached.

Compared to content-based reputation systems, address-based reputation systems are much more popular. Among them, blacklists and whitelists are the simplest. Blacklists only contain the identity of spammers, while whitelists only record the identity of legitimate senders. E-mail address based whitelists and blacklists, for example, DOEmail [28], are commonly used by individuals. However, countless new spam addresses and spoofed legitimate addresses render blacklists almost useless in practice and spam with spoofed whitelisted addresses can avoid filtering and contaminate whitelists. To defeat e-mail address spoofing, many sender authentication schemes have been proposed, in which SPF (Sender Policy Framework) [109] and DKIM (DomainKeys Identified Mail) [14] are the most noticeable. SPF and DKIM can help identify the sending party but cannot determine its legitimacy as spammers also embrace these schemes [82, 94]. As one

type of IP-address-based blacklists, DNSBLs (e.g., [4, 5]) disseminate blacklists through DNS and are widely used. They only contain the IP addresses of spamming sources on centrally-managed servers. DNSBLs detect spamming hosts by using either e-mail traps (or called honeypots) or by end-user contribution. However, the effectiveness of DNSBLs has been questioned [27, 72, 73]. Besides DNSBLs, other types of IP-address-based reputation systems such as [3] and [2] also exist. These systems usually are commercial and use proprietary techniques for reputation maintenance and dissemination.

The Gmail reputation system [94] rates domains instead of IP addresses. It uses only local information and identifies the sending domain using both heuristics and SPF and DKIM. Singaraju *et al.* [81] proposed a collaborative e-mail reputation framework called RepuScore, which also rates domains. RepuScore relies on a central authority to collect information from collaborative domains and manage the reputation database.

Besides qualitative rating approaches, a few quantitative rating methods have been proposed. Leiba *et al.* [50] presented an algorithm to derive the reputation of e-mail domains and IP addresses by analyzing the SMTP sending paths (in the message header) of known legitimate messages and spam messages. The reputation score for each IP address is based on the number of spam and non-spam messages which contain the address in their sending paths. Each intermediate address in a sending path is associated with a credit value to prevent spammers from forging the sending path. Within the context of e-mail social networks, Golbeck *et al.* [33] proposed an algorithm to infer the relative reputation ratings (the reputation of a sender may be different in the eyes of different recipients) of e-mail contacts based on the exchange of reputation values. Chirita *et al.* [25] developed a reputation scheme called MailRank, which can compute a global reputation score as well as a personalized score for each e-mail address. MailRank uses pre-selected e-mail

addresses with high reputation in bootstrap, and the rating computation is biased towards the pre-determined address set to combat spammer attacks. However, both systems mentioned above do not cope with address spoofing attacks. Both [33] and [25] assume the existence of global e-mail social networks and compute reputation scores in a centralized manner.

Collaboration has been applied into the spam signature generation and e-mail address whitelist population. To expand whitelists in an automatic manner, LOAF [24], FOAF [21], and RE: [30] have been developed. These systems leverage the social connections, specifically the friend-of-a-friend relations among people, to find indirect relations between senders and recipients.

5.2 Motivation

As demonstrated by [98] and [101], local e-mail histories can be used to enhance the quality of e-mail service. However, they also reveal that it is impossible to cover all incoming e-mail messages merely based on local history. In other words, there are always messages from unseen sources. Unfortunately, the dynamics of such messages, which directly affect the performance of a local-history-based reputation system, have not yet been studied. This motivated our following study on the dynamics of incoming e-mail.

We collected 151-day e-mail logs for inbound messages from our campus e-mail servers. The logs are daily-based and span from 2007/11/01 to 2008/03/31 with only one daily log missing. For each inbound message, we logged the time of message arrival, the IP address and domain name (if any) of the sending server, and the spamminess score (between 0 and 300, the bigger, the more likely to be spam) given by the spam filter. We removed those records without valid fully qualified domain names (FQDN), since their corresponding messages are almost certainly spam. As original logs do not contain name of the domain each sending server resides in, we derived the

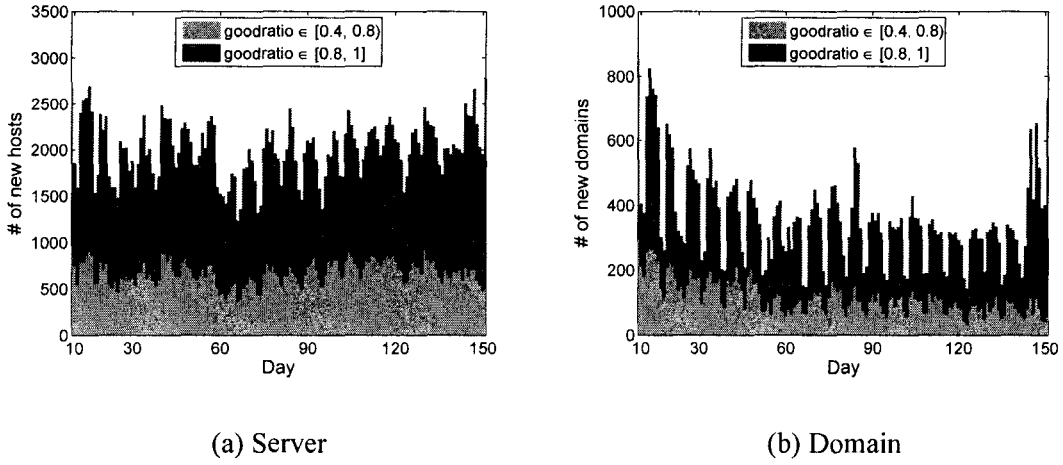


Figure 5.1: Number of newly-appeared senders per day

domain information using `dig` and added it to the log. We observed that a significant number of newly-appeared (never recorded by any previous logs) servers and domains consistently show up in daily logs, even after 100 days. The average numbers of newly-appeared servers and domains per day are 27,733 and 1,152, respectively. More importantly, this observation also holds for those servers and domains that mostly send legitimate e-mail.

We use metric “good-ratio” to measure sending behavior of a server (and domain). The good-ratio of a sending server/domain is computed by dividing the number of nonspam messages over the number of total messages sent from the server/domain across all logs. Good-ratio “1” means always sending nonspam e-mail and good-ratio “0” means always sending spam. To reduce the false positives (misclassified nonspam messages) caused by the spam filter, we set spam threshold as 10 (the default threshold set by the spam filter is 50) and classify the messages with spamminess scores less than or equal to 10 as nonspam and the rest as spam. This classification results in zero false positive and about 1.8% false negatives (uncaught spam) in one of campus e-mail archives, which contains 1,800 manually verified spam messages. We further reduce

false negatives by applying a few well-established heuristics, which identify sending servers with dynamically-allocated IP addresses by their domain names. For example, home PCs with broadband connections such as cable modem and DSL mostly have dynamically-allocated IP addresses and their domain names usually either contain keywords such as “cable” and “dsl” or have distinct patterns of number and alphabet combinations such as “ip10-23-45-67”. Because hosts with dynamically-allocated IP addresses are widely exploited by spammers and the messages from those hosts are almost absolutely spam, we mark the messages from dynamically-allocated IP addresses as spam and change their spamminess scores to 100. Despite that false positives and false negatives may still exist after conservative classification and rectification, we believe the misclassification is minor and should not affect our measurement results.

Figure 5.1 illustrates the dynamics of the numbers of newly-appeared servers and domains whose good-ratios are no less than 0.4 in daily logs. The servers (domains) are further divided into two groups; one group with good-ratio in $[0.4, 0.8)$ and the other with good-ratio in $[0.8, 1]$. In Figure 5.1, we can see that the number of newly-appeared servers (domains) per day is not negligible. For example, even after 100 days, newly-appeared servers with good-ratio over 0.8 per day are still counted by thousands. Compared to newly-appeared servers per day, newly-appeared domains with high good-ratios per day are counted by hundreds, still too many to be ignored.

These measurement results suggest that using only local history information may not be sufficient for building a high-quality reputation system. Intuitively, the coverage of senders can be improved through collaboration, as an e-mail sender that is new to one receiver may be old to others. Naturally, the peers that have frequent e-mail communications and behave consistently well become candidates of collaboration. As shown in one of recent spam studies [101], there exist good e-mail servers from which most of e-mail is nonspam for most of time. However, that

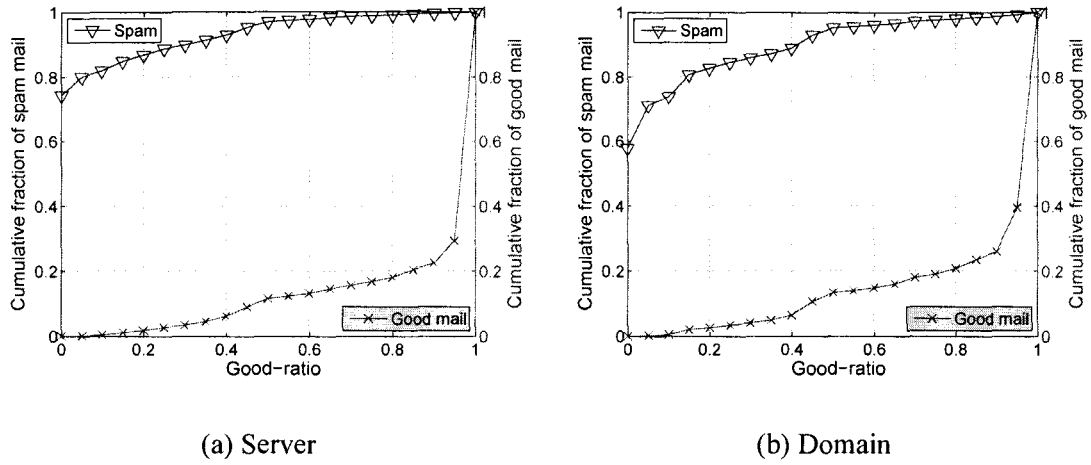


Figure 5.2: CDF of good-ratio for spam and nonspam (good) e-mail

analysis is based on the data from one vantage point and does not study the sender behavior at the domain level. Therefore, we use our e-mail logs to verify if their observation holds here and up to the domain level.

Taking spam (nonspam) messages from the servers with valid domain names in all logs as a whole, we examine the proportion of spam (nonspam) contributed by the servers with a specific good-ratio. we plot the CDFs (cumulative distribution functions) of good-ratio for spam and nonspam at host level in Figure 5.2 (a). The curve in the left top shows the CDF for spam while the curve in the right bottom shows the CDF of nonspam. The CDFs at the domain level are shown in Figure 5.2 (b). We also examined the CDFs with different time window (number of days) and time range (starting and ending days) and found that those CDFs are very similar to those shown in Figure 5.2. In general, our results conform to the findings in [101]. The servers with high good-ratios send the majority of nonspam e-mail and the servers with low good-ratios send the most of spam, which makes the use of reputation system very helpful. For instance, the servers with good-ratios no smaller than 0.8 send over 80% of total nonspam e-mail but less than 1% of total

spam. Hence, whitelisting these servers would save a great deal of filtering resources and improve the delivery of legitimate e-mail. Moreover, the server-level observations also apply at the domain level. The curve shapes in Figure 5.2 (b) are very similar to those in Figure 5.2 (a), indicating that well-behaved domains do exist.

Based on the measurement results, we conclude that (1) e-mail senders can be rated by their long-term behaviors; (2) local observation may not suffice for building a high-quality rating system. These two factors are instrumental to the design of CARE.

5.3 System Design

CARE is designed to be an autonomous system. Each domain equipped with CARE independently determines collaborating domains, exchanges information with collaborating domains, and derives the reputation scores of remote domains. Information exchange occurs between two domains that mutually agree on collaboration. In case no collaboration is available, the system can continue functioning by using only local e-mail history information. The autonomy eases incremental deployment of CARE.

As a reputation system, CARE operates collaboratively with other anti-spam techniques. A typical use of CARE is functioning as a preprocessor of a content-based spam filter. Messages from domains with sufficiently high reputation scores are directly accepted while messages from domains with very low reputation scores are directly rejected. For the rest of e-mail, those messages from domains with no reputation are directly passed to the filter and all the others are marked with their reputation scores before passing.

The architecture of CARE is shown in Figure 5.3. The local e-mail history module takes the log of local e-mail servers as input and derives the local Database of E-mail History (DEH). The

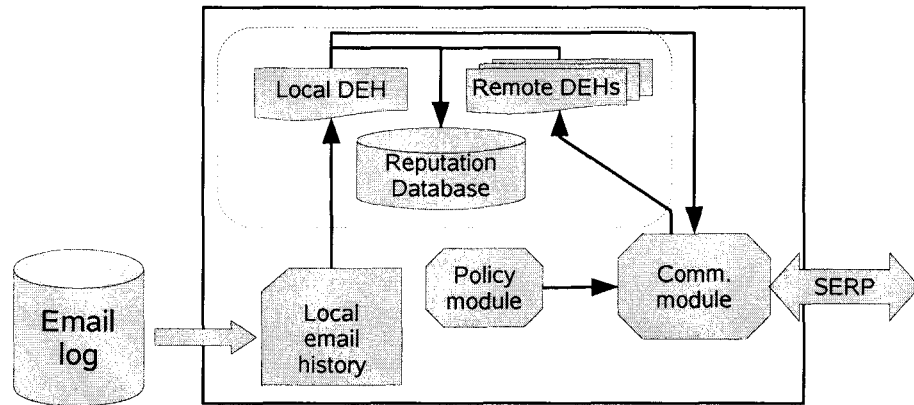


Figure 5.3: Architecture of CARE system

information of local DEH is used in both information exchange and reputation derivation. In information exchange, the collaborating domains are determined based on their behaviors recorded by local DEH. In reputation derivation, the local DEH involves in both calculating reputation scores and assessing the trustworthiness of remote information from collaborating domains. Through the communication module, a CARE system exchanges information with other collaborating systems via Simple E-mail Reputation Protocol (SERP). To provide flexibility in deciding collaborating domains, a policy module is incorporated allowing system administrators to apply their admission control policies. For example, system administrators may install policies into the policy module to forbid local domain to share information with certain remote domains. Using both local DEH and remote DEHs, reputation scores of e-mail sending domains are computed and stored into the reputation database.

In this section, we first highlight the rating issues in CARE. Then, we describe how to build the local e-mail history and reputation database. After that, we detail the communication module and SERP protocol.

5.3.1 Rating Issues

CARE uses domain as the reputation identity, following Gmail reputation service [94] and RepuScore [81]. Rating domain is preferred to rating server (IP address), due to the following reasons. First, domain is easier to be authenticated than IP address, as e-mail authentication schemes (e.g., SPF and DKIM) have become popular. Second, e-mail sending policies are usually applied at domain level for nonspam domains and rating domain can provide better scalability. Third, an IP address can be used by multiple entities simultaneously, while a domain represents only one entity at any time. Last but not least, legitimate e-mail servers are usually placed in a separate subdomain in a large ISP and can be easily distinguished from the subdomain where a spam botnet resides.

CARE does not differentiate e-mail relaying servers from e-mail originating servers in rating. If an e-mail server not only sends e-mail for its own domain, but also relays e-mail for other domains, all relayed messages will be counted onto the relay domain. “No open-relay” has been a rule for e-mail server administration and well followed by decent e-mail service providers.

5.3.2 Local E-mail History

Local e-mail history contains the information of both inbound and outbound e-mail transmissions occurred locally. If multiple e-mail servers are used in the local domain, the local e-mail history is the integration of the information recorded by all those servers. Local e-mail history records the basic information of e-mail transmissions such as e-mail transfer time, spam or not, source and destination at the host and domain levels. This information usually can be directly extracted from e-mail log. The domain of a remote host can be easily retrieved, e.g., by using the DNS utility “dig”. The local e-mail history does not include the messages in e-mail log that have no corresponding domains. In general legitimate e-mail servers of an ISP can be distinguished

from spam bots residing inside the same ISP by domain name, because legitimate e-mail servers are usually placed in a separate domain for management and security reasons. For instance, the broadband host “ip70-161-245-78.hr.hr.cox.net” is in domain “hr.cox.net” while the legitimate e-mail server “smtp.west.cox.net” is in domain “west.cox.net”. E-mail authentication schemes such as SPF and DKIM can further enhance the accuracy of domain identification as the binding between an e-mail server and its domain is explicitly expressed by special DNS records. The local e-mail history can be updated either online or offline.

A CARE system also maintains a special database called Database of E-mail History (DEH), which is derived from the local e-mail history and used in information sharing. Each sending domain has a record $(\mathcal{X}, TM, GM, AD)$ in the database. A record profiles the e-mail sending behavior of a domain in the past W days. W is tunable parameter and decided by the system administrator. \mathcal{X} is the name of sending domain. TM and GM are the numbers of total messages and good (i.e., nonspam) messages from \mathcal{X} , respectively. AD is the number of the active days, in each of which, at least one message from \mathcal{X} is received. The database is updated periodically, e.g., twice per day, and the history information beyond W days could be removed to save disk space.

5.3.3 Reputation Database

We derive a domain’s reputation by combining both local DEH and the remote DEHs collected from collaborating domains. Initially, we only have local database. Under this circumstance, the reputation derivation is simplified into computing the good-ratio of each domain in the local database. After exchanging information with collaborating domains, we also use remote databases in derivation of domain reputation. However, the information from collaborating domains may not

Algorithm 2 Computing Domain Reputation

- 1: Input: $\text{DEH}_{\mathcal{J}}$ and all collected remote $\text{DEH}_{\mathcal{R}}$ s.
 - 2: Output: reputation score for every sender in $\text{DEH}_{\mathcal{J}}$ and $\text{DEH}_{\mathcal{R}}$ s.
 - 3: **for** each remote $\text{DEH}_{\mathcal{R}}$ **do**
 - 4: compute trustworthiness score $\theta_{\mathcal{R}}$.
 - 5: **end for**
 - 6: **for** each sender \mathcal{X} in $\text{DEH}_{\mathcal{J}}$ and those $\text{DEH}_{\mathcal{R}}$ s that contain it **do**
 - 7: compute the reputation score of \mathcal{X} .
 - 8: **end for**
-

be fully trusted, because the authenticity of information is self-warranted. Therefore, we introduce a trustworthiness score for each remote database. In the absence of a central authority, we rely on the local database to assess the trustworthiness of a remote database. Specifically, we examine the correlation between the local database and a remote database on sending domains in common, and use the correlation result to compute the trustworthiness score of that database. The remote databases with high trustworthiness scores are deemed reliable. A domain's reputation score is the weighted average of good-ratios derived from the local and remote databases, and the weight for each remote database is the trustworthiness score of the database.

Algorithm 2 describes the general process of domain reputation computing. The notations used in the algorithm and the rest of the section are summarized in Table 5.1. In general, the subscript of a symbol represents a history recording domain; it also can represent the domain's DEH when the context is clear. We use \mathcal{R} for a generic collaborating domain and \mathcal{J} for the local domain. The superscript of a symbol represents an e-mail sending domain (seen by either local domain or a collaborating domain). For clarity, a collaborating domain with which we exchange

Table 5.1: Summary of notations

DEH	Database of E-mail History
SMD	Set of Major Domains, for history correlation
W	History Window
\mathcal{X}	Domain \mathcal{X}
$KU(\mathcal{X})$	public key of domain \mathcal{X}
$KI(\mathcal{X})$	private key of domain \mathcal{X}
$GM_{\mathcal{D}}^{\mathcal{X}}$	number of good messages from \mathcal{X} received by \mathcal{D}
$TM_{\mathcal{D}}^{\mathcal{X}}$	number of total messages from \mathcal{X} received by \mathcal{D}
$AD_{\mathcal{D}}^{\mathcal{X}}$	number of active days \mathcal{X} sending e-mail to \mathcal{D}
$dg_{\mathcal{D}}^{\mathcal{X}}$	good-ratio of \mathcal{X} according to $DEH_{\mathcal{D}}$
$ds_{\mathcal{D}}^{\mathcal{X}}$	domain score of \mathcal{X} according to $DEH_{\mathcal{D}}$
$dr^{\mathcal{X}}$	domain reputation of \mathcal{X}
$\gamma_{\mathcal{R}}$	supporting factor of $DEH_{\mathcal{R}}$ for computing $\theta_{\mathcal{R}}$
$\omega_{\mathcal{R}}$	correlation coefficient of $DEH_{\mathcal{R}}$ for computing $\theta_{\mathcal{R}}$
$\theta_{\mathcal{R}}$	trustworthiness score (weight) of $DEH_{\mathcal{R}}$
β	threshold used in constructing SMD
δ	threshold used in computing $\gamma_{\mathcal{R}}$

information is termed as a collaborator, while a domain logged in either local history or a remote history is termed as a sender.

The process of computing domain reputation consists of two steps. In the first step (lines 3–5) we compute the trustworthiness score of each remote database $DEH_{\mathcal{R}}$, and in the second step

(lines 6–8) we compute the reputation score of each sender recorded by either local database or remote databases.

To compute the trustworthiness score of remote database $\text{DEH}_{\mathcal{R}}$, we first derive the Set of Major Domains (SMD) of that database. SMD contains those domains that behave well and stay active. Such a well-behaved domain is indicated by a large domain score. The domain score is defined as $ds = \frac{GM}{TM} \times \frac{AD}{W}$. $ds_{\mathcal{X}}$ stands for the domain score of sender \mathcal{X} in database $\text{DEH}_{\mathcal{R}}$ and can be easily computed from the record of \mathcal{X} in $\text{DEH}_{\mathcal{R}}$. Senders with a sufficiently large domain score are added into SMD, that is, $\text{SMD} = \{\mathcal{X} : ds_{\mathcal{X}} \geq \beta\}$, where β is the threshold decided locally. By setting β to an appropriate value, which implies a high good-ratio (we define the good-ratio as $dg = \frac{GM}{TM}$) and a high ratio of active days ($\frac{AD}{W}$), we can ensure that the majority of domains in SMD are legitimate.

Then, we compute the intersection set (denoted by INT) between local SMD ($\text{SMD}_{\mathcal{S}}$) and \mathcal{R} 's SMD ($\text{SMD}_{\mathcal{R}}$) for remote database $\text{DEH}_{\mathcal{R}}$. Formally, $\text{INT}_{\mathcal{R}} = \text{SMD}_{\mathcal{S}} \cap \text{SMD}_{\mathcal{R}}$. We also compute supporting factor $\gamma_{\mathcal{R}}$ from $\text{INT}_{\mathcal{R}}$. By definition,

$$\gamma_{\mathcal{R}} = \frac{\min(|\text{INT}_{\mathcal{R}}|, \delta)}{\delta}, \quad (5.1)$$

where $|S|$ represents the cardinality of set S , and δ is a pre-defined system parameter. The rationale behind computing SMD and INT is to find a reliable common base for correlation computing. In other words, the sending domains in common for correlation computing (i.e., the set of domains represented by INT) are expected to present consistent sending behaviors to receiving domains including the local domain \mathcal{S} and remote domain \mathcal{R} . Legitimate e-mail service providers usually present this characteristic. Computing γ is to take the size of common base into consideration.

After that, we compute the correlation coefficient of $\text{DEH}_{\mathcal{R}}$ (denoted by $\omega_{\mathcal{R}}$) from $\text{INT}_{\mathcal{R}}$

based on city block distance (also called Manhattan distance, taxicab distance). For the domains in $\text{INT}_{\mathcal{R}}$, we first derive their good-ratio vectors in $\text{DEH}_{\mathcal{J}}$ and $\text{DEH}_{\mathcal{R}}$ (denoted as $\mathbf{V}_{\mathcal{J}}$ and $\mathbf{V}_{\mathcal{R}}$ respectively). With $\text{INT}_{\mathcal{R}} = \{\mathcal{X}^1 \dots \mathcal{X}^n\}$, $\mathbf{V}_{\mathcal{J}} = [dg_{\mathcal{J}}^{\mathcal{X}^1} \dots dg_{\mathcal{J}}^{\mathcal{X}^n}]$ and $\mathbf{V}_{\mathcal{R}} = [dg_{\mathcal{R}}^{\mathcal{X}^1} \dots dg_{\mathcal{R}}^{\mathcal{X}^n}]$. Then, the city block distance between $\mathbf{V}_{\mathcal{J}}$ and $\mathbf{V}_{\mathcal{R}}$, denoted as $\text{dist}(\mathbf{V}_{\mathcal{J}}, \mathbf{V}_{\mathcal{R}})$, is obtained by summing up the difference of good-ratios in $\text{DEH}_{\mathcal{J}}$ and $\text{DEH}_{\mathcal{R}}$ for each domain in $\text{INT}_{\mathcal{R}}$. Formally,

$$\text{dist}(\mathbf{V}_{\mathcal{J}}, \mathbf{V}_{\mathcal{R}}) = \sum_{i=1}^n |dg_{\mathcal{J}}^{\mathcal{X}^i} - dg_{\mathcal{R}}^{\mathcal{X}^i}|. \quad (5.2)$$

We get the correlation coefficient $\omega_{\mathcal{R}}$ by normalizing $\text{dist}(\mathbf{V}_{\mathcal{J}}, \mathbf{V}_{\mathcal{R}})$ into $[0, 1]$, that is,

$$\omega_{\mathcal{R}} = 1 - \frac{\text{dist}(\mathbf{V}_{\mathcal{J}}, \mathbf{V}_{\mathcal{R}})}{||\text{INT}_{\mathcal{R}}||}. \quad (5.3)$$

We derive the trustworthiness score of $\text{DEH}_{\mathcal{R}}$, $\theta_{\mathcal{R}}$, by multiplying $\text{DEH}_{\mathcal{R}}$'s supporting factor $\gamma_{\mathcal{R}}$ and its correlation coefficient $\omega_{\mathcal{R}}$. Formally,

$$\theta_{\mathcal{R}} = \gamma_{\mathcal{R}} \cdot \omega_{\mathcal{R}} = \frac{\min(||\text{INT}_{\mathcal{R}}||, \delta)}{\delta} \cdot (1 - \frac{\text{dist}(\mathbf{V}_{\mathcal{J}}, \mathbf{V}_{\mathcal{R}})}{||\text{INT}_{\mathcal{R}}||}). \quad (5.4)$$

We also incorporate a list of trusted collaborators into CARE. Administrators can add their trusted collaborating domains into the list or remove any domain from it. The weight of each domain in the list, that is, θ , is 1. This offsets the potential inaccuracy in computing the trustworthiness score, since it is possible that a collaborating domain's view is different from the local view on the same sending domain.

Finally, for each domain \mathcal{X} we derive its reputation score $dr^{\mathcal{X}}$ by computing the weighted average of \mathcal{X} 's good-ratios ($dg^{\mathcal{X}}$) from the DEHs that record \mathcal{X} . We use \mathcal{D} to represent a generic domain whose DEH contains a record for \mathcal{X} . The weight for remote database $\text{DEH}_{\mathcal{R}}$, $\theta_{\mathcal{R}}$, is in

$[0, 1]$ and the weight for the local database $\text{DEH}_{\mathcal{D}}$, $\theta_{\mathcal{D}}$, is always 1. Formally, the reputation score of domain \mathcal{X} is defined as

$$dr^{\mathcal{X}} = \frac{\sum_{\mathcal{D} \in Q} \theta_{\mathcal{D}} \cdot dg_{\mathcal{D}}^{\mathcal{X}}}{\sum_{\mathcal{D} \in Q} \theta_{\mathcal{D}}}, \quad (5.5)$$

where $Q = \{\mathcal{D} : \mathcal{X} \in \text{DEH}_{\mathcal{D}}\}$. Note that the local good-ratio $dg_{\mathcal{D}}^{\mathcal{X}}$ can be 0. In this case, sender \mathcal{X} has not been recorded by the local domain. By using the weighted average, the sending domains recorded by both the local domain and collaborators are assessed from a broader view, while the sending domains recorded only by the local domain are not affected.

5.3.4 Simple E-mail Reputation Protocol (SERP)

CARE systems communicate with one another via SERP. Through SERP, a CARE system can transfer DEH as well as other messages to its counterpart. SERP adopts a DNS-based authentication scheme, borrowing the idea of DNS-based e-mail authentication schemes. The DNS-based authentication is lightweight, easy to install, and incrementally deployable. SERP requires every deployment domain (e.g., example.com) to publish a special TXT resource record¹ in its `_care` DNS subdomain (`_care.example.com` in this example). The record must specify the domain name (or IP address) of the host on which CARE is serviced and the associated public key. By doing so, a remote CARE host can be authenticated by first querying the TXT DNS record under the `_care` subdomain of the domain where the host resides, and then checking if the domain name (or IP address) of the host is listed in that record.

Among all domains that have direct e-mail communication with the local domain, CARE selects those domains that behave consistently well for collaboration. These domains can be easily

¹In case DNS SRV resource record [36] is chosen to publish CARE service, a separate DNS TXT record is still needed for carrying public key and CARE host information.

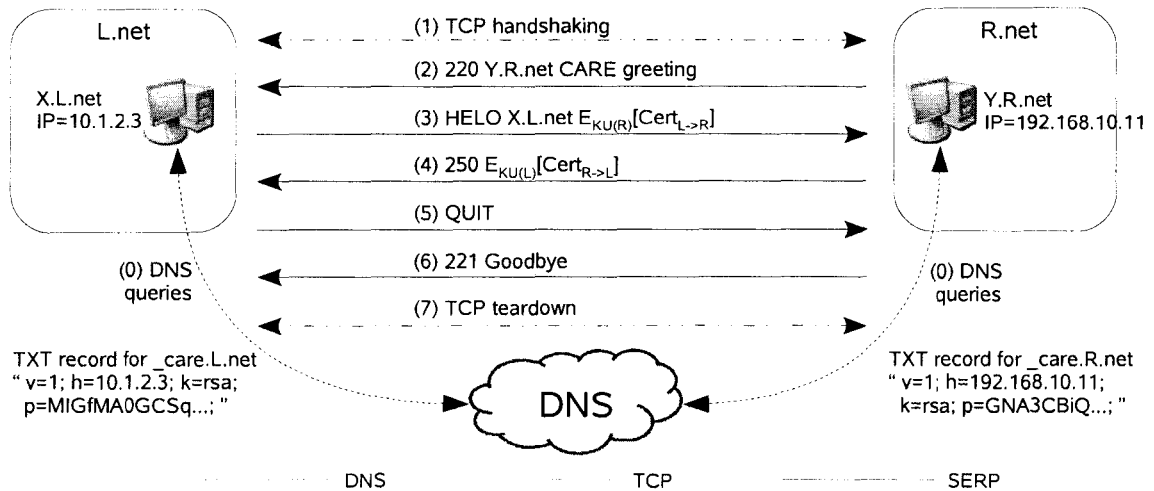


Figure 5.4: Procedure of a successful mutual agreement establishment via SERP

decided by examining the local e-mail history. Apparently, they also must have a valid TXT DNS record for CARE. Each CARE host maintains a list of remote domains satisfying these requirements and uses it for selecting collaborating domains.

To collaborate, two domains are required to reach a mutual agreement on information exchange before sharing DEHs. With the agreement, the two domains will play dual roles of service requester and provider.

Figure 5.4 illustrates the procedure of establishing a successful mutual agreement via SERP. In the figure, the CARE hosts in domain L.net (L for short) and R.net are X.L.net (X for short) and Y.R.net, respectively. Both domains have published their CARE DNS records. Since every CARE system keeps a list of collaborating domains, by periodically querying the CARE DNS records of those domains, each system can readily know the positions of CARE hosts inside those domains. The activity of periodic DNS query is shown as step (0) in Figure 5.4. After X successfully resolves the domain name of Y, it sends a request to Y for establishing a TCP connection. When Y receives this request, by checking its list of collaborating domains, it can instantly decide how

to react: accepting the request if the remote host is in the list or rejecting the request otherwise. In this example, Y accepts the request. The TCP handshaking process is marked as step (1) in Figure 5.4. Semantically, neither step (0) nor step (1) is part of SERP. However, SERP needs step (0) for authentication and step (1) for a reliable data connection and authorization.

After the TCP connection is established, host Y sends a greeting message to the requesting host X (step (2)), indicating its identity. After receiving the greeting message, X issues command “HELO” (step (3)), followed by the domain name of X and a certificate ($\text{Cert}_{L \rightarrow R}$) encrypted by the public key of domain R (i.e., $KU(R)$). The certificate $\text{Cert}_{L \rightarrow R}$ means that domain L allows domain R to access L’s DEH. It is composed by concatenating message M and its signature, that is, $\text{Cert}_{L \rightarrow R} = M || E_{KI(L)}[H(M)]$, where $H(M)$ is the hash value of M and $KI(L)$ is the private key of domain L. Message M contains: certificate issuer L and recipient R, certificate starting and expiration times, and the updating interval of L’s DEH. The communication proceeds if the certificate is accepted by Y. Echoing the offer of X, Y responds by sending its certificate $\text{Cert}_{R \rightarrow L}$ back to X (step (4)). After a successful exchange, X sends command “QUIT” (step (5)), indicating completion of the mutual agreement. The TCP connection is torn down (step (7)) as soon as Y acknowledges the “QUIT” command (step (6)). The above procedure will be repeated once either of the certificates expires.

After mutual agreement, two domains can exchange their DEHs with each other. The data exchange procedure is similar to the agreement establishing procedure. The data exchange can be optimized since DEH is usually updated gradually. We can make a snapshot of DEH as the reference base and generate a difference file for each update to DEH. Then, we just transfer the appropriate difference file(s) instead of a whole DEH, reducing the total bytes of data transmission.

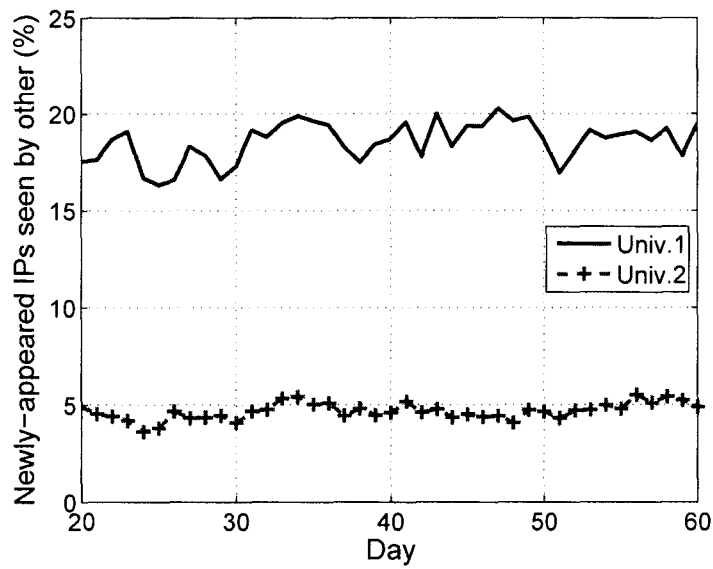


Figure 5.5: Percentage of newly-appeared IP addresses that have been recorded by the other university over all newly-appeared IP addresses in daily logs

5.4 System Evaluation

Our evaluation focuses on the potential benefit of using CARE. Specifically, we are interested in the increase of coverage brought by collaboration, that is, the reduction of the number of newly-appeared sending domains thanks to collaboration. We have analyzed real e-mail logs, conducted a real-world estimating experiment, and performed extensive simulations to validate the effectiveness of CARE on improving the coverage. In addition, we apply a set of simulations to demonstrate the cheating-resistance of CARE.

5.4.1 Log-based Experiment

We first collected two-month e-mail logs from two universities recorded in the same time period. All e-mail logs are daily based and record the source IP addresses (no domain information) and

spam information for inbound messages.

For newly-appeared IP addresses in the daily logs of each university, we examine how many of them have already been recorded by the other university and calculate the percentage. The dynamics of the percentage in the daily logs are shown in Figure 5.5, which demonstrates the effectiveness of collaboration. The curve for university 1 shows that about 16% to 20% of the newly-appeared IP addresses in university 1's daily logs have already been recorded in university 2's logs. For university 2, the percentage reduces to 5% but is pretty stable. The difference of percentage for two universities may be attributed to the difference of total IP addresses in their e-mail logs. On average, university 1 records 42,850 IP addresses daily, while university 2 observes 87,815 IP addresses in a day. However, the stability of both curves implies the stable gain from collaboration in the long run.

5.4.2 DNS-based Experiment

Results from this log comparison are encouraging. However, due to access limits, we cannot obtain more e-mail logs for evaluating CARE comprehensively. To estimate the potential benefit that could be achieved by multi-domain collaboration, we applied DNS snooping. Due to the fact that a DNS MX query is usually made before an SMTP transaction, by exploiting the DNS caching mechanism, we can approximately infer whether e-mail has been sent to a given domain by snooping (using iterative mode) the DNS cache of the sending domain. If the MX record of the receiving domain can be found in the DNS cache, it is highly likely that e-mail communication between the two domains occurred recently. Clearly, the number of cache hits by DNS snooping may not accurately reflect real e-mail communications. Nevertheless, DNS snooping provides an efficient way of estimating the gain of multi-domain collaboration and the derived result can serve

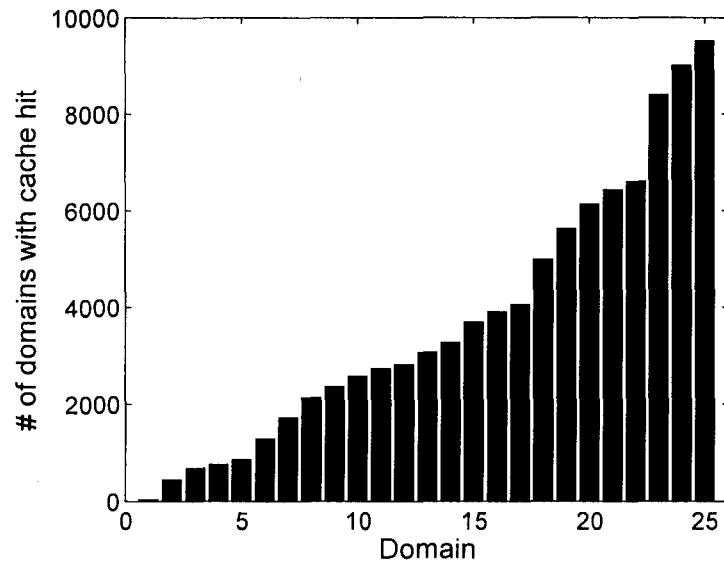


Figure 5.6: Number of DNS cache hits for 25 .edu domains

as a lower bound.

We randomly selected 25 .edu domains from the logs used in Section 5.2 as collaborating domains, and randomly selected 50,000 *inactive* domains as DNS snooping target domains. Each of these snooped domains sent less than 5 messages in total in the logs, and none of them had sent any messages in the past month before snooping. We use these inactive domains as “ongoing” and “new” sending domains to study how many of them can be covered by our collaborators.

Not all inactive domains can be snooped. After DNS probing test, 36,646 out of 50,000 domains were used as probing targets. Then, we probed the DNS servers of those domains to find out how many MX records of the selected 25 .edu domains can be hit. Finally, we group probe results by each edu domain and show the number of probed domains that had cache hits for the target edu domain in Figure 5.6. From the figure, we can see an obvious diversity on the number of probed domains with a cache hit among different edu domains. Some edu domains can be hit

in DNS caches of more than 8,000 inactive domains, while some other edu domains have less than 1,000 hits. For an edu domain, a hit in the DNS cache of an inactive domain means that the edu domain received e-mail from the inactive domain and thus had this domain in its local e-mail history. Therefore, we can benefit more by collaborating with the edu domains that have more cache hits. Overall, the total number of cache hits for 25 .edu domains is 12,660, indicating that the e-mail history from 25 collaborating .edu domains can cover at least 34.6% of newly-appeared domains. The gain from multi-domain collaboration could be higher with more collaborators and more types (e.g., .com) of collaborators. In addition, we probed all the inactive domains within one day. This implies that all 36,646 domains appeared in the same day, which, however, is unlikely to happen in practice according to our measurement results. Thus, the benefit could be even higher in reality.

5.4.3 Simulation Experiments

We applied simulation to further study the dynamic characteristics and cheating-resistance of CARE. We implemented a CARE simulator with full CARE functionality. The simulator is driven by the configuration of e-mail domains (1,200 nonspam domains and 10,000 spam domains) and the daily traces of e-mail communications among those domains (60 days). Both spam and non-spam domains are dynamically born in the trace. Nonspam domains stay until the end of trace, while spam domains only stay for a short random period. Spam domains always send spam to non-spam domains, while nonspam domains send to one another both nonspam and spam messages. We use three types of network topologies (power-law, small world, and random graph) for non-spam domains. We readily acknowledge that the generated traces may not be representative, since there is no a priori knowledge on the topology and dynamics of e-mail domains. However, the em-

phasis here is on the effectiveness of CARE with no assumption on network and communication patterns.

The simulator first randomly picks a given percentage of nonspam domains as CARE domains using the domain configuration, and then starts simulation using the daily traces. In each day, the simulator first does the message receiving and history recording driven by the trace records, then updates the reputation database of each CARE domain. In simulation, CARE is used as the preprocessor of a spam filter. The spam filter has fixed false positive rate and false negative rate, 0.01 and 0.05, respectively. Messages from a domain with reputation score 0.8 or higher are regarded as nonspam and messages from a domain with reputation score 0.1 or less are regarded as spam. If the reputation score cannot ensure acceptance or rejection, CARE tags the message with no-drop and passes it to the filter. All processing results are logged into the database of e-mail history to compute reputation. All CARE domains use the same parameter setting (history window $W = 30$, $\beta = 0.3$, and $\delta = 3$).

We first investigate how CARE improves domain coverage through collaboration. Figure 5.7 shows the dynamics of percentage of newly-appeared nonspam domains that are covered after using CARE in each day. The results are displayed from day 20 because of history accumulation. The “net1”, “net2”, and “net3” stand for power-law topology, small world topology, and random graph topology, respectively. To illustrate the effect of increasing CARE deployment rate on the coverage, we set the percentage of the nonspam domains that use CARE as 10%, 20%, and 30% in the simulation, and display their results in Figure 5.7. For each given combination of network setting and CARE domain percentage, we run ten trials and use the average as the result.

Figure 5.7 clearly demonstrates the effectiveness of CARE collaboration on improving the coverage of nonspam domains. Moreover, we can see that the increase of percentage of CARE

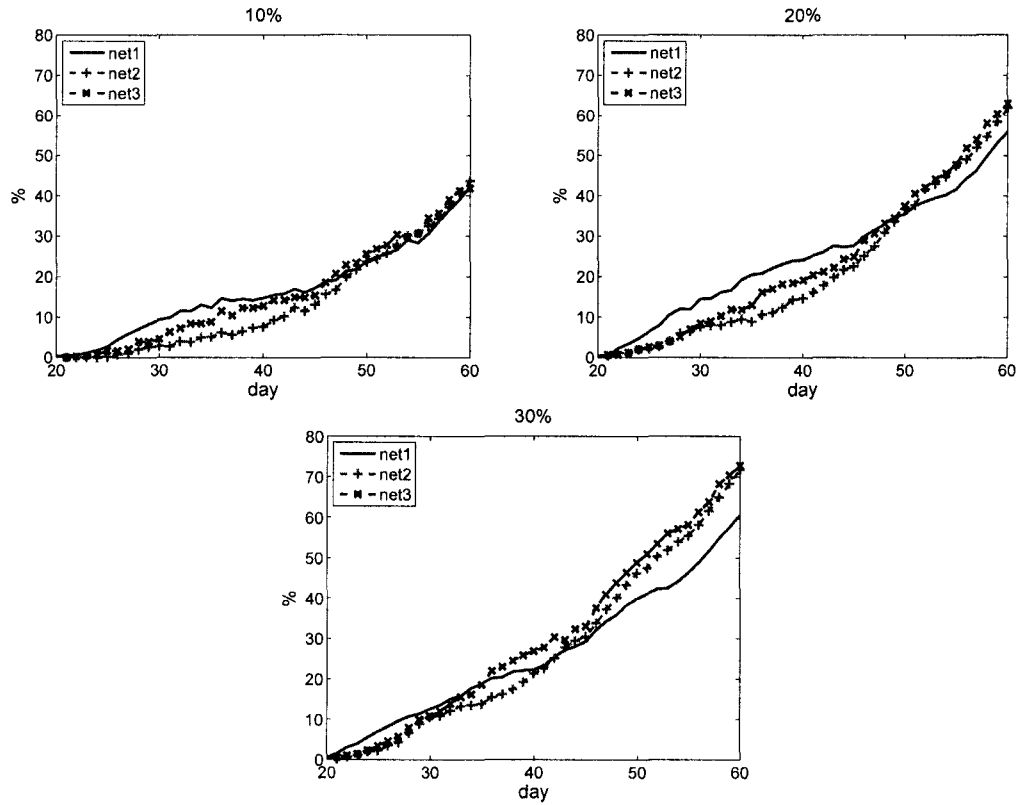


Figure 5.7: Percentage of newly-appeared nonspam domains that are covered after using CARE in each day

deployments renders the growth of percentage of domains being covered. Figure 5.8 shows the dynamics of percentage of newly-appeared spam domains that are covered after using CARE in each day. The observations from Figure 5.7 also hold in Figure 5.8. Therefore, CARE can effectively improve the coverage for both spam domains and nonspam domains.

Bigger coverage indicates that more nonspam messages may have reputation scores and be protected from being dropped by spam filter. We compare the number of the nonspam messages that are directly accepted under CARE collaboration to that under no CARE collaboration and find that CARE does increase the number of directly accepted nonspam messages. We show the percentage of increase in terms of nonspam messages being directly accepted in each day in Figure

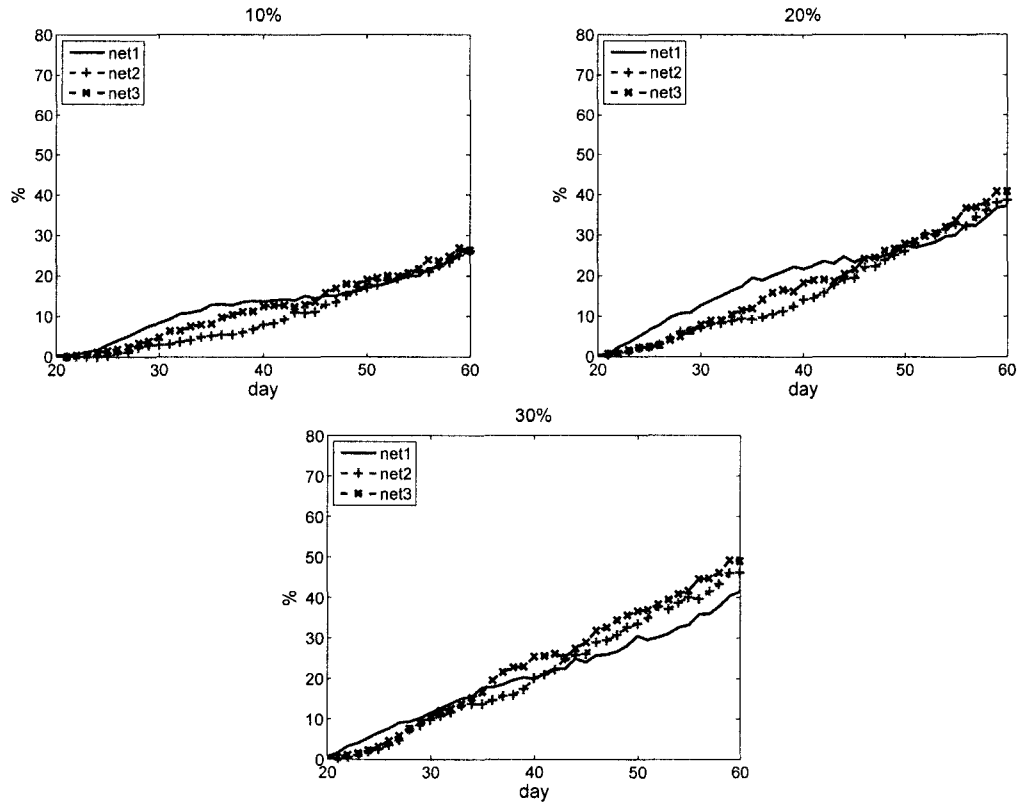


Figure 5.8: Percentage of newly-appeared spam domains that are covered after using CARE in each day

5.9. We can see that the acceptance of more nonspam messages, with all three different network topologies, has been achieved and that the percentage of increase keeps growing with time in any of the simulation environments. This indicates that the use of CARE can prevent considerably more nonspam messages from being lost. It is also notable from Figure 5.9 that more CARE deployments (10% vs. 20%) result in more nonspam messages being directly accepted.

Bigger coverage also means that more spam messages may be identified early and directly rejected. Our comparison of the the spam messages that are directly rejected under CARE collaboration with that under no CARE collaboration reveals the benefit of using CARE on fighting spam. The percentage of increase in terms of spam messages being directly rejected in each day

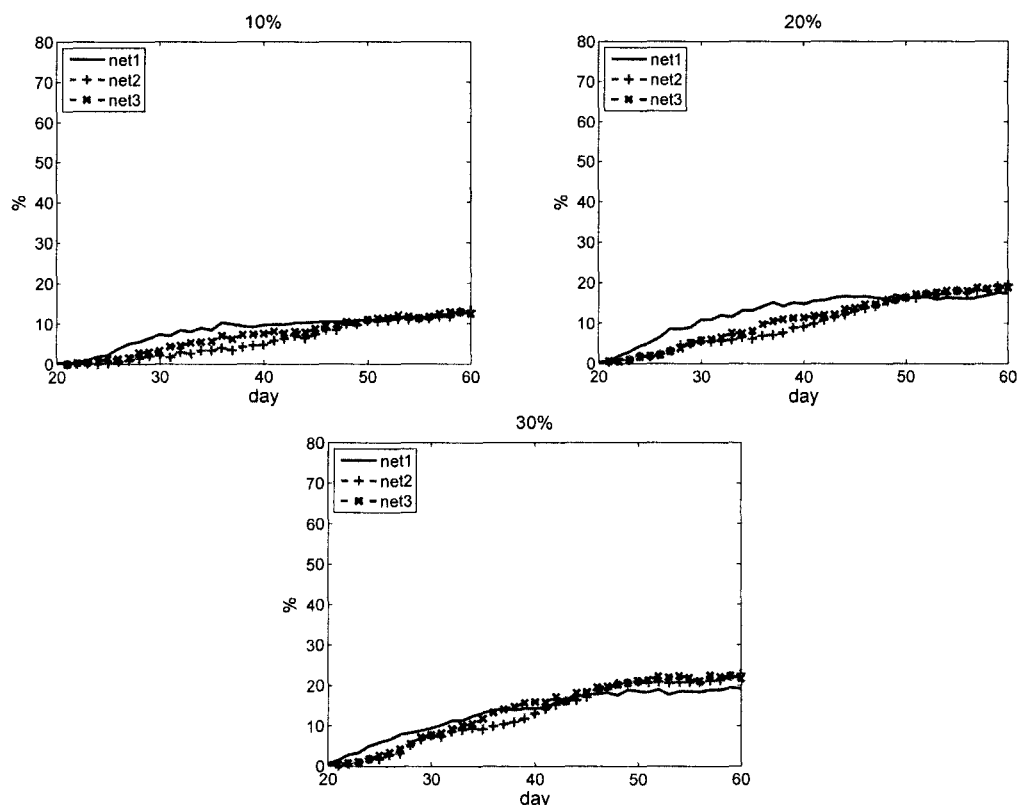


Figure 5.9: Percentage of more nonspam messages being directly accepted after using CARE in each day

is shown in Figure 5.10. Clearly, more spam messages can be rejected when more CARE systems are deployed and their usage becomes longer.

We are also interested in the cheating-resistance of CARE. Although it is difficult for a spammer to cheat other domains by mimicking good behavior, it is relatively easy for a nonspam domain with high reputation to tamper its own database and cheat its collaborators. To study this type of cheating, we randomly pick 10% of CARE domains as cheating domains and rerun the simulation. A cheating domain always sends a forged database to its collaborators in information exchange. The forged database has the same records as the genuine database except that the value of GM is changed to $TM - GM$, that is, the complement of original GM . We find that CARE can

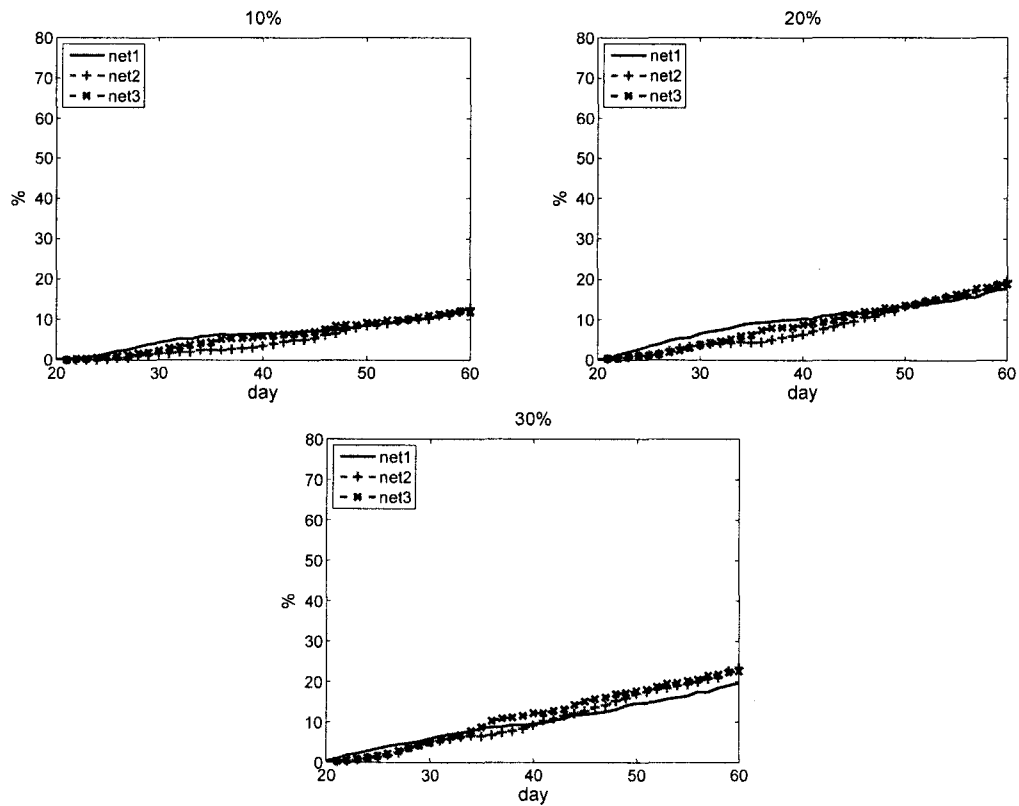


Figure 5.10: Percentage of more spam messages being directly rejected after using CARE in each day

effectively resist this type of attacks. For those messages being directly rejected, there is no false positive no matter the cheating scheme is applied or not; For those messages being directly accepted, the difference of false negatives with and without cheating is negligible. Cheating causes only about 0.1% reduction in the total number of messages being accepted, rejected, or tagged by CARE when 30% of nonspam domains use CARE. In general, the simulation shows that the studied cheating scheme, when applied at a small scale, has negligible effect on CARE.

5.5 Summary

In this chapter we have presented the motivation, design, and evaluation of CARE, a collaboration-based autonomous e-mail reputation system. CARE is a generic e-mail reputation system in that it rates both spam and nonspam domains in an autonomous manner. Using CARE, each domain derives the reputation scores of e-mail sending domains by sharing its local observations with those domains that manifest consistently good behavior. In the evaluation of CARE, we used two large e-mail log traces from two universities to quantify the benefits of collaboration between two domains and conducted a large DNS snooping experiment to estimate the potential gain brought by multi-domain collaboration. Moreover, we performed extensive simulations to further reveal the dynamics and attack-resistance of CARE in a large-scale environment. Our experimental results evidently demonstrate the effectiveness of CARE.

Chapter 6

Conclusions and Future Work

This dissertation focuses on securing Internet-based message systems, a critically important yet considerably difficult undertaking for today's modern society. To achieve the goal, we identify that the following two complementary tasks must be fulfilled. They are (1) defending against unwanted messages and (2) ensuring delivery of wanted messages. For the first task, we have developed the DBSpam system to thwart e-mail spam laundering and the HoneyIM system to counter malicious instant messages. For the second task, we have designed the CARE e-mail reputation system to improve the reliability of e-mail services.

To fight against e-mail spam laundering, one of major approaches to pumping e-mail spam, we have thoroughly studied the laundering mechanisms and distilled the unique characteristics of connection correlation and packet symmetry from the behavior of spam laundering. We have developed a simple yet effective technique, DBSpam, to online detect and break spam laundering activities inside a customer network based on the packet symmetry. We have implemented a prototype of DBSpam using *libpcap* on Linux and extensively evaluated its effectiveness and performance through trace-based experiments. The experimental results demonstrate that DBSpam

is effective in quickly and accurately capturing and suppressing e-mail spam laundering activities and is capable of coping with high speed network traffic.

We have developed a generic framework, HoneyIM, to foil the spread of malicious instant messages in an enterprise-like network. We exploit the inherent characteristic of IM malware spreading and apply the honeypot technology to the detection of malicious instant messages. Specifically, HoneyIM uses decoy accounts in normal users' contact lists as honeypots to capture malicious messages sent by IM malware and suppresses the spread of malicious instant messages by performing network-wide blocking. We have implemented a prototype of HoneyIM for public IM services, based on open-source IM client Pidgin and client honeypot Capture. We have validated the efficacy of HoneyIM through both simulations and real experiments.

To largely improve e-mail reliability, that is, reduce losses of wanted e-mail, we have designed a collaboration-based autonomous e-mail reputation system called CARE. CARE introduces inter-domain collaboration without central authority or third party and enables each e-mail service provider to independently build its reputation database, including both frequently contacted and unacquainted sending domains, using the local e-mail history and the information exchanged with other collaborating domains. We have conducted a number of experiments to validate the effectiveness of CARE on improving e-mail reliability, including comparing two large e-mail log traces from two universities, conducting a real experiment of DNS snooping on more than 36,000 domains, and performing extensive simulation experiments in a large-scale environment.

6.1 Future Work

There are a couple of enhancements and extensions of our existing research that we would like to explore in the future. First, we would like to incorporate more anomaly detection techniques into

HoneyIM for deep inspection purpose and evaluate their accuracy and performance in practice more comprehensively. New types of IM malware continuously emerge and some of those may be specially designed to evade certain detections. Therefore, different types of deep inspection techniques are needed. How to combine different detection results and derive accurate information about suspicious instant messages would be an interesting problem for our future research.

Second, we would like to explore the applicability of HoneyIM to enterprise e-mail systems. As the trend of fusing different message systems, especially e-mail and instant messaging, is surfacing in enterprises and organizations, it would be natural and desirable to apply a uniform and comprehensive protection scheme for both e-mail and instant messaging in the environments that have integrated message systems. To that end, HoneyIM is expected to counter not only malicious instant messages and e-mail messages, but also unwanted bogus advertisements, scam and phishing messages, etc. Therefore, it would be necessary to transform HoneyIM into a general message protection system that incorporate both anti-e-mail-spam techniques and anti-IM-malware techniques. We plan to investigate how to build such a system and make it both effective and efficient.

We have designed CARE and evaluated its effectiveness using simulations. In the future, we plan to implement CARE as the open source plug-ins, which contain statistics report functionality, for mainstream spam filters such as SpamAssassin [90]. We aim to distribute them and collect practical data to evaluate the effectiveness of CARE in reality.

Last but not least, we plan to leverage the power of GPU (Graphics Processing Unit) to improve the performance of content-based spam filters. Content-based spam filters are very popular. However, they have relatively high computational cost and may not scale well with the increase of message volume. To improve the performance of spam filters while keeping their simple single-threaded architecture, the performance bottleneck of spam filtering should be offloaded. As GPUs

have been powerful and cost-effective, we plan to employ GPUs in the computationally intensive tasks and develop a GPU-based generic message processing engine to support different content-based filtering systems.

Bibliography

- [1] The growth of e-mail spam. http://en.wikipedia.org/wiki/E-mail_spam#The_growth_of_e-mail_spam. [Accessed: Oct. 26, 2009].
- [2] Habeas senderindex. <http://www.habeas.com/Services/For-Receiver/SenderIndex/>.
- [3] Sender score. <http://www.senderscore.org/>.
- [4] SpamCop Blocking List. <http://www.spamcop.net/bl.shtml>.
- [5] The Spamhaus Project. <http://www.spamhaus.org/>.
- [6] Top layer security. <http://www.toplayer.com>.
- [7] Silent email loss by earthlink. http://www.pbs.org/cringely/pulpit/2006/pulpit_20061201_001274.html, 2006.
- [8] Pidgin. <http://pidgin.im/>, 2007.
- [9] Skype worm outbreak highlights importance of proactive protection. <http://sg.hardwarezone.com/news/view.php?id=8562&cid=8>, 2007. [Accessed: Oct. 26, 2009].
- [10] Storm worm. http://en.wikipedia.org/wiki/Storm_Worm, 2007.
- [11] The radicati group, inc. releases email statistics report, 2009-2013. <http://www.reuters.com/article/pressRelease/idUS119688+06-May-2009+MW20090506>, 2009.
- [12] Total online activities. <http://www.pewinternet.org/Static-Pages/Trend-Data/Online-Activites-Total.aspx>, 2009.
- [13] SHARAD AGARWAL, VENKATA N. PADMANABHAN, AND DILIP A. JOSEPH. Addressing email loss with suremail: Measurement, design, and evaluation. In *Proc. USENIX Annual Technical Conference 2007*, pages 281–294, Santa Clara, CA, June 2007.
- [14] E. ALLMAN, J. CALLAS, M. DELANY, M. LIBBEY, J. FENTON, AND M. THOMAS. RFC 4871: Domainkeys identified mail (DKIM) signatures, May 2007.
- [15] DMITRI ALPEROVITCH, PAUL JUDGE, AND SVEN KRASSER. Taxonomy of email reputation systems. In *ICDCS 2007 Workshops*, page 27, Toronto, Canada, June 2007.

- [16] MAURO ANDREOLINI, ALESSANDRO BULGARELLI, MICHELE COLAJANNI, AND FRANCESCA MAZZONI. Honeyspam: Honeypots fighting spam at the source. In *Proceedings of the 1st USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, pages 77–83, Cambridge, MA, July 2005.
- [17] PAUL BÄCHER, THORSTEN HOLZ, MARKUS KÖTTER, AND GEORG WICHESKI. Know your enemy: Tracking botnets. <http://www.honeynet.org/papers/bots/>, March 2005.
- [18] ADAM BACK. Hashcash - a denial of service counter-measure. <http://www.hashcash.org/papers/hashcash.pdf>, April 1997.
- [19] JEREMY BLOSSER AND DAVID JOSEPHSEN. Scalable centralized bayesian spam mitigation with bogofilter. In *Proceedings of the 18th USENIX Systems Administration Conference (LISA)*, pages 1–20, Atlanta, GA, November 2004.
- [20] AVRIM BLUM, DAWN XIAODONG SONG, AND SHOBHA VENKATARAMAN. Detection of interactive stepping stones: Algorithms and confidence bounds. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sophia Antipolis, France, September 2004.
- [21] DAN BRICKLEY AND LIBBY MILLER. FOAF vocabulary specification 0.9. <http://xmlns.com/foaf/spec/>, 2007.
- [22] TIAN BU AND DON TOWSLEY. On Distinguishing Between Internet Power Law Topology Generators. In *Proceedings of the 2002 IEEE INFOCOM*, pages 638–647, New York, NY, June 2002.
- [23] CBL. Composite blocking list. <http://cbl.abuseat.org>, 2007.
- [24] MACIEJ CEGLOWSKI AND JOSHUA SCHACHTER. LOAF. <http://loaf.cantbedone.org/>, 2004.
- [25] PAUL-ALEXANDRU CHIRITA, JORG DIEDERICH, AND WOLFGANG NEJDL. MailRank: Using ranking for spam detection. In *Proceedings of ACM International Conference on Information and Knowledge Management*, pages 373–380, Bremen, Germany, October 2005.
- [26] MARK DELANY. Internet draft: Domain-based email authentication using public keys advertised in the DNS (DomainKeys), July 2006.
- [27] ZHENHAI DUAN, KARTIK GOPALAN, AND XIN YUAN. Behavioral characteristics of spammers and their network reachability properties. In *Proc. IEEE ICC 2007*, Glasgow, Scotland, June 2007.
- [28] DAVID ERICKSON. DOEmail - default off email. <http://doemail.org/>.
- [29] R. FIELDING, J. GETTYS, J. MOGUL, H. FRYSTYK, L. MASINTER, P. LEACH, AND T. BERNERS-LEE. RFC 2616: Hypertext transfer protocol – http/1.1, June 1999.

- [30] SCOTT GARRISS, MICHAEL KAMINSKY, MICHAEL J. FREEDMAN, BRAD KARP, DAVID MAZIERES, AND HAIFENG YU. Re: Reliable email. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 297–310, San Jose, CA, May 2006.
- [31] PAWEŁ GBURZYŃSKI AND JACEK MAITAN. Fighting the spam wars: A remailer approach with restrictive aliasing. *ACM Transactions on Internet Technology*, 4(1):1–30, February 2004.
- [32] RANDALL GELLENS AND JOHN C. KLENSIN. RFC 2476: Message submission, December 1998.
- [33] JENNIFER GOLBECK AND JAMES HENDLER. Reputation network analysis for email filtering. In *CEAS 2004*, Mountain View, CA, July 2004.
- [34] ALEKS GOSTEV. Social engineering: the latest chapter. <http://www.viruslist.com/en/weblog?weblogid=168136245>, August 2005.
- [35] PAUL GRAHAM. A plan for spam. <http://www.paulgraham.com/spam.html>, 2002.
- [36] A. GULBRANDSEN, P. VIXIE, AND L. ESIBOV. RFC 2782: A DNS RR for specifying the location of services (DNS SRV), 2000.
- [37] SHLOMO HERSHKOP AND SALVATORE J. STOLFO. Combining email models for false positive reduction. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 98–107, Chicago, IL, August 2005.
- [38] MATTHEW HICKS. Reuters suspends im service due to kelvir worm. <http://www.eweek.com/article2/0,1759,1786151,00.asp>, April 2005.
- [39] NEAL HINDOCHA AND ERIC CHIEN. Malicious Threats and Vulnerabilities in Instant Messaging. <http://www.symantec.com/avcenter/reference/malicious.threats.instant.messaging.pdf>, 2003.
- [40] TIM HUNTER, PAUL TERRY, AND ALAN JUDGE. Distributed tarpitting: Impeding spam across multiple servers. In *Proceedings of the 17th USENIX Systems Administration Conference (LISA)*, pages 223–236, San Diego, CA, October 2003.
- [41] IBM. Lotus Sametime. <http://www-142.ibm.com/software/sw-lotus/sametime>.
- [42] JOHN IOANNIDIS. Fighting spam by encapsulating policy in email addresses. In *Proceedings of the 10th Annual Network and Distributed System Security Symposium (NDSS)*, pages 1–8, San Diego, CA, February 2003.
- [43] JAEYEON JUNG, VERN PAXSON, ARTHUR W. BERGER, AND HARI BALAKRISHNAN. Fast portscan detection using sequential hypothesis testing. In *Proceedings of the 25th IEEE Symposium on Security and Privacy*, pages 211–225, Oakland, CA, May 2004.
- [44] JAEYEON JUNG AND EMIL SIT. An empirical study of spam traffic and the use of DNS black lists. In *Proceedings of ACM SIGCOMM Internet Measurement Conference 2004*, pages 370–375, Taormina, Italy, October 2004.

- [45] KASPERSKY. IM-Worm.Win32.Bropia.aj. <http://www.viruslist.com/en/viruses/encyclopedia?virusid=72841>.
- [46] KASPERSKY. IM-Worm.Win32.Opanki.d. <http://www.viruslist.com/en/viruses/encyclopedia?virusid=84950>.
- [47] J. KLENSIN. RFC 2821: Simple mail transfer protocol, April 2001.
- [48] BALACHANDER KRISHNAMURTHY AND ED BLACKMOND. SHRED: Spam harassment reduction via economic disincentives. <http://www.research.att.com/~bala/papers/shred-ext.pdf>, 2004.
- [49] M. LEECH, M. GANIS, Y. LEE, R. KURIS, D. KOBLAS, AND L. JONES. RFC 1928: Socks protocol version 5, March 1996.
- [50] BARRY LEIBA, JOEL OSSHER, V.T. RAJAN, RICHARD SEGAL, AND MARK WEGMAN. Smtip path analysis. In *CEAS 2005*, Mountain View, CA, July 2005.
- [51] KANG LI, CALTON PU, AND MUSTAQUE AHAMAD. Resisting spam delivery by tcp damping. In *Proceedings of the 1st Conference on Email and Anti-Spam*, pages 191–198, Mountain View, CA, July 2004.
- [52] KANG LI AND ZHENYU ZHONG. Fast statistical spam filter by approximate classifications. In *Proceedings of ACM 2006 SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 347–358, St. Malo, France, June 2006.
- [53] ZHIJUN LIU AND DAVID LEE. Coping with instant messaging worms - statistical modeling and analysis. In *Proceedings of the 15th IEEE Workshop on Local and Metropolitan Area Networks*, Princeton, NJ, June 2007.
- [54] ZHIJUN LIU, GUOQIANG SHU, NA LI, , AND DAVID LEE. Defending against instant messaging worms. In *Proceedings of IEEE GLOBECOM 2006*, pages 1–6, San Francisco, CA, Nov. 2006.
- [55] JIM LYON AND MENG WENG WONG. Internet draft: Sender id: Authenticating e-mail, October 2004.
- [56] MOHAMMAD MANNAN AND PAUL C. VAN OORSCHOT. Secure Public Instant Messaging: A Survey. In *Proceedings of the 2nd Annual Conference on Privacy, Security, and Trust*, pages 69–77, Fredericton, NB, Canada, 2004.
- [57] MOHAMMAD MANNAN AND PAUL C. VAN OORSCHOT. On Instant Messaging Worms, Analysis and Countermeasures. In *Proceedings of WORM 2005*, pages 2–11, Fairfax, VA, Nov. 2005.
- [58] MARID. MTA authorization records in DNS. <http://www.ietf.org/html.charters/OLD/marid-charter.html>, 2004.
- [59] MICROSOFT. Office Live Communications Server. <http://www.microsoft.com/office/livecomm/prodinfo/default.mspx>.

- [60] MICROSOFT. The penny black project. <http://research.microsoft.com/research/sv/PennyBlack/>, 2003.
- [61] CHERYL D. MORSE AND HAINING WANG. The Structure of An Instant Messenger Network and Its Vulnerability to Malicious Codes. In *Proceedings of ACM SIGCOMM 2005 Poster Session*, Philadelphia, PA, Aug. 2005.
- [62] NETFILTER. iptables project. <http://www.netfilter.org/projects/iptables/>.
- [63] JAMES NEWSOME AND DAWN SONG. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of the 12th NDSS*, San Diego, CA, Feb. 2005.
- [64] NORMAN. Norman sandbox whitepaper. http://download.norman.no/whitepapers/whitepaper_Norman_SandBox.pdf.
- [65] GEORGIOS PORTOKALIDIS, ASIA SLOWINSKA, AND HERBERT BOS. Argos: an emulator for fingerprinting zeroday attacks. In *Proceedings of the EUROSYS 2006*, Leuven, Belgium, April 2006.
- [66] POSTINI. Sender behavior analysis. <http://www.postini.com>, 2006.
- [67] VIPUL VED PRAKASH AND ADAM O'DONNELL. Fighting spam with reputation systems. *ACM Queue*, 3(9), November 2005.
- [68] NIELS PROVOS. A virtual honeypot framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, San Diego, CA, August 2004.
- [69] SVETLANA RADOSAVAC, JOHN S. BARAS, AND IORDANIS KOUTSOPOULOS. A framework for mac protocol misbehavior detection in wireless networks. In *Proceedings of the 4th ACM workshop on Wireless security (WiSe)*, pages 33–42, Cologne, Germany, September 2005.
- [70] COSTIN RAIU. The IM worms armada. <http://www.viruslist.com/en/weblog?weblogid=203678309>, October 2006.
- [71] ANIRUDH RAMACHANDRAN, DAVID DAGON, AND NICK FEAMSTER. Can dns-based blacklists keep up with bots? In *CEAS 2006*, Mountain View, CA, July 2006.
- [72] ANIRUDH RAMACHANDRAN AND NICK FEAMSTER. Understanding the network-level behavior of spammers. In *Proc. ACM SIGCOMM 2006*, Pisa, Italy, September 2006.
- [73] ANIRUDH RAMACHANDRAN, NICK FEAMSTER, AND SANTOSH VEMPALA. Filtering spam with behavioral blacklisting. In *Proc. ACM CCS 2007*, Alexandria, VA, October 2007.
- [74] RHYOLITE. Distributed checksum clearinghouse (dcc). <http://www.rhyolite.com/anti-spam/dcc/>, 2000.
- [75] MARTIN ROESCH. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Systems Administration Conference (LISA)*, pages 229–238, Seattle, WA, November 1999.

- [76] MIKE SACHOFF. Botnets driving spam volume. <http://www.webpronews.com/topnews/2009/07/29/botnets-driving-spam-volume>, 2009. [Accessed: Oct. 26, 2009].
- [77] ROEL SCHOUWENBERG. Kelvir changes its approach. <http://www.viruslist.com/en/weblog?weblogid=162243612>, April 2005.
- [78] ROEL SCHOUWENBERG. Do you like photos? <http://www.viruslist.com/en/weblog?weblogid=199354341>, Sept. 2006.
- [79] ROEL SCHOUWENBERG. MSN filter bypassing - part 2. <http://www.viruslist.com/en/weblog?weblogid=199850358>, Sept. 2006.
- [80] SECURITYTRACKER. Formmail.pl web-to-email cgi script allows unauthorized users to send mail anonymously. <http://www.securitytracker.com/alerts/2001/Mar/1001108.html>, 2001.
- [81] GAUTAM SINGARAJU AND BYUNGHOON KANG. Repuscore: Collaborative reputation management framework for email infrastructure. In *Proc. the 21st Large Installation System Administration (LISA)*, 2007.
- [82] GAUTAM SINGARAJU, JEFF MOSS, AND BYUNGHOON KANG. Tracking email reputation for authenticated sender identities. In *CEAS 2008*, 2008.
- [83] REGINALD D. SMITH. Instant Messaging as a Scale-Free Network. <http://arxiv.org/abs/cond-mat/0206378v2>, 2002.
- [84] SOPHOS. Troj/Kelvir-M. <http://www.sophos.com/virusinfo/analyses/trojkelvirm.html>.
- [85] SOPHOS. W32/Jitux-A. <http://www.sophos.com/virusinfo/analyses/w32jituxa.html>.
- [86] SOPHOS. W32/Kelvir-F. <http://www.sophos.com/virusinfo/analyses/w32kelvirf.html>.
- [87] SOPHOS. W32/Kelvir-Q. <http://www.sophos.com/virusinfo/analyses/w32kelvirq.html>.
- [88] SOPHOS. W32/Sohana-R. <http://www.sophos.com/security/analyses/w32sohanar.html>.
- [89] SORBS. Spam and open relay blocking system (sorbs). <http://www.sorbs.net/>, 2006.
- [90] SPAMASSASSIN. The apache spamassassin project. <http://spamassassin.apache.org/>, 2006.
- [91] SPAMHAUS. Increasing spam threat from proxy hijackers. <http://www.spamhaus.org/news.lasso?article=156>, 2005.
- [92] SPAMLINKS. Challenge/response spam filters. <http://spamlinks.net/filter-cr.htm>, 2006.

- [93] RAMON STEENSON AND CHRISTIAN SEIFERT. Capture: A high interaction client honeypot. <http://www.nz-honeynet.org/capture.html>.
- [94] BRADLEY TAYLOR. Sender reputation in a large webmail service. In *CEAS 2006*, Mountain View, CA, 2006.
- [95] THE HONEYNET PROJECT. *Know Your Enemy: Learning about Security Threats (2nd Edition)*. Addison-Wesley Professional, May 2004.
- [96] AARJAV J. TRIVEDI, PAUL Q. JUDGE, AND SVEN KRASSER. Analyzing Network and Content Characteristics of Spim Using Honeypots. In *Proceedings of the 3rd USENIX SRUTI*, Santa Clara, CA, June 2007.
- [97] AARON TURNER. Tcpreplay. <http://tcpreplay.synfin.net/trac/>, 2006.
- [98] RICHARD DANIEL TWINING, MATTHEW M. WILLIAMSON, MIRANDA MOWBRAY, AND MAHER RAHMOUNI. Email prioritization: Reducing delays on legitimate mail caused by junk mail. In *Proc. USENIX Annual Technical Conference 2004*, 2004.
- [99] RICHARD DANIEL TWINING, MATTHEW M. WILLIAMSON, MIRANDA MOWBRAY, AND MAHER RAHMOUNI. Email prioritization: Reducing delays on legitimate mail caused by junk mail. In *Proceedings of USENIX 2004 Annual Technical Conference*, pages 45–58, Boston, MA, June 2004.
- [100] JAMIE TWYXCROSS AND MATTHEW M. WILLIAMSON. Implementing and testing a virus throttle. In *Proceedings of the 12th USENIX Security Symposium*, pages 285–294, Washington, DC, August 2003.
- [101] SHOBHA VENKATARAMAN, SUBHABRATA SEN, OLIVER SPATSCHECK, PATRICK HAFFNER, AND DAWN SONG. Exploiting network structure for proactive spam mitigation. In *Proc. USENIX Security 2007*, page 149166, Boston, MA, August 2007.
- [102] ABRAHAM WALD. *Sequential Analysis*. Dover Publications, June 2004.
- [103] MICHAEL WALFISH, J.D. ZAMFIRESCU, HARI BALAKRISHNAN, DAVID KARGER, AND SCOTT SHENKER. Distributed quota enforcement for spam control. In *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pages 281–296, San Jose, CA, May 2006.
- [104] YI-MIN WANG, DOUG BECK, XUXIAN JIANG, ROUSSE ROUSSEV, CHAD VERBOWSKI, SHUO CHEN, AND SAM KING. Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities. In *Proceedings of the 13th NDSS*, San Diego, CA, Feb. 2006.
- [105] DAVID WATSON, THORSTEN HOLZ, AND SVEN MUELLER. Know your enemy: Phishing. <http://www.honeynet.org/papers/phishing/>, May 2005.
- [106] MICHAEL WELZL. end2end-interest: A message to authors of pfldnet papers. <http://mailman.postel.org/pipermail/end2end-interest/2008-January/>, 2008.

- [107] MATTHEW M. WILLIAMSON. Design, implementation and test of an email virus throttle. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 76–85, Las Vegas, Nevada, December 2003.
- [108] MATTHEW M. WILLIAMSON, ALAN PARRY, AND ANDREW BYDE. Virus throttling for instant messaging. Technical report, HP Lab Bristol, May 2004.
- [109] MENG WENG WONG AND WAYNE SCHLITT. RFC 4408: Sender policy framework (SPF) for authorizing use of domains in e-mail, version 1, April 2006.
- [110] DALE WOOLRIDGE, JAMES LAW, AND MOTO KAWASAKI. The qmail spam throttle mechanism. <http://spamthrottle.qmail.ca/man/qmail-spamthrottle.5.html>, 2004.
- [111] ZHEN XIAO, LEI GUO, AND JOHN TRACEY. Understanding Instant Messaging Traffic Characteristics. In *Proceedings of the 27th ICDCS*, Toronto, Canada, June 2007.
- [112] BILL YERAZUNIS. CRM114 - the controllable regex mutilator. <http://crm114.sourceforge.net>, 2003.
- [113] YIN ZHANG AND VERN PAXSON. Detecting stepping stones. In *Proceedings of the 9th USENIX Security Symposium*, pages 171–184, Denver, CO, August 2000.
- [114] FENG ZHOU, LI ZHUANG, BEN Y. ZHAO, LING HUANG, ANTHONY D. JOSEPH, AND JOHN KUBIATOWICZ. Approximate object location and spam filtering on peer-to-peer systems. In *Proceedings of the 4th ACM/IFIP/USENIX International Middleware Conference*, Markus Endler and Douglas Schmidt, editors, volume 2672 of *LNCS*, pages 1–20, Rio de Janeiro, Brazil, June 2003. Springer Berlin/Heidelberg.
- [115] CLIFF C. ZOU, DON TOWSLEY, AND WEIBO GONG. Modeling and Simulation Study of the Propagation and Defense of Internet Email Worm. *IEEE Transactions on Dependable and Secure Computing*, 4(2):105–118, April-June 2007.

VITA

Mengjun Xie

Mengjun Xie received his Bachelor of Engineering and Master of Engineering degrees, both in Computer Science, from the East China Normal University in 1999 and 2002 respectively. He has been a PhD student in the Department of Computer Science at the College of William and Mary since 2003. His research interests lie in network security, network systems, system security, and operating systems. His current focus is on message systems security and reliability.