Privacy-Preserving Distributed, Automated Signature-Based Detection of New Internet Worms

Hyang-Ah Kim

May 2010
CMU-CS-10-122

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
David R. O'Hallaron, Co-Chair
Dawn Song, Co-Chair
Brad Karp, University College London
Vern Paxson, ICSI

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

This work is sponsored in part by the National Science Foundation under Grants CNS-0509004 and IIS-0429334, in part by a subcontract from the Southern California Earthquake Center's CME Project as part of NSF ITR EAR-01-22464, and in part by support from the Intel Corporation, and Carnegie Mellon CyLab.

Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of NSF, the Intel Corporation, Carnegie Mellon CyLab, or other funding parties.

Keywords: Internet Worm Containment, Worm Signature Generation, Content Prevalence Analysis, Privacy-Preserving Collaboration, Distributed Monitoring

Abstract

This dissertation develops techniques, based on monitoring network traffic, that automate signature generation for wide-spreading malicious payloads such as Internet worms. Fast signature detection is required to achieve effective content-based filtering. The main thesis is that content prevalence analysis in network payloads across distributed networks is a good basis for automated signature generation for wide-spreading malicious payloads, and can be performed without compromising the privacy of participating networks.

Content-prevalence analysis extracts unique payload patterns that are identical and invariant over all the flows that convey a wide-spreading malicious payload. Distributed monitoring enables us to rapidly capture many sample payloads, thus expediting the signature generation. Extra care for privacy encourages more networks to participate in the distributed monitoring and makes the approach practical.

The first part of this dissertation presents a system, *Autograph*, that generates network payload signatures for Internet worms by utilizing the content invariance and wide-spreading communication patterns of Internet worm traffic. Signature generation speed is improved further by extending Autograph to share port scanner lists with distributed Autograph monitors. Trace-driven simulation shows the fundamental trade-off between early generation of signatures for novel worms and specificity of the generated signatures.

Distributed monitoring is a recognized technique in security to expedite worm detection. However, extra care for privacy must be taken. The second part of the dissertation presents two techniques for privacy-preserving distributed signature generation. HotItemID protects the data and owner privacy by using sampling techniques and hiding private data in a crowd. Another technique protects privacy using privacy-preserving multiset operation framework. The technique relies on a semantically secure homomorphic cryptosystem and arithmetic operations over polynomial representation of sets. Both techniques protect privacy based on the assumption that a payload appearing in multiple locations should not be private. The dissertation confirms the assumption by studying real network traffic traces, and shows that privacy-preserving distributed worm signature detection is feasible.

Acknowledgments

I am deeply grateful to my advisors, David R. O'Hallaron and Dawn Song. David has offered advice on which classes to take, how to do research, write papers, and give talks, interact with collegues, and hack systems. He was also my English teacher. He provided me much more freedom than any student could ask for, but was always there to listen, help, and encourage whenever I was lost and needed his help, and never gave up on me. Dawn has provided many constuctive feedbacks and advices on what to read and how to approach problems. Her enthusiasic and energetic attitute, and breadth of knowledge in security, cryptography, and privacy, was impressive and she was one of my role models in my graduate school life.

I would like to thank Brad Karp, who served as my unofficial advisor and later my thesis committee member. I met Brad when I did internship in Intel Research Pittsburgh. He was my advisor in Intel and we developed Autograph together. He guided me through the design and implementation of the system from defining the problem, getting resources, to writing papers, giving presentations, and releasing the code of Autograph as open-source.

I would also like to thank my thesis committee member, Vern Paxson for his feedback and support. Much of his work in network-based intrusion detection, measurement, and worm analysis were the foundations of my work. I deeply appreciated his detailed and constructive comments on my thesis.

I would also like to thank professors in Carnegie Mellon, Michael Reiter, Hui Zhang, Srinivasan Seshan, Peter Steenkiste, Satya, Mor Harchol-Balter, and Garth Gibson, for their valuable advice which was not limited only on research. I would like to thank all of my colleagues and coauthors, Yinglian Xie, Lea Kissner, and those who often attended network seminars, listened to my talks, and gave many valuable comments.

I am grateful to Vern Paxon of ICSI, Sue Moon and her students of KAIST, Yong Lee and Hyong Kim of KISA and Cylab-Korea, Casey Helfrich and James Gurganus of Intel Research Pittsburgh, for their support that granted me the access to traces and analysis infrastructure, and for valuable feedback on my work. Without their support, I couldn't write this thesis.

Many people made my years at Carnegie Mellon enjoyable and memorable. I'd like to thank all of my friends and classmates, including but not limited to: Hisun Kim, Yejong Kim, Keywon Chong, Sungwoo Park, Sehyun Yang, Mukesh Agrawal, Maverick Wu, Soyoung Park, Rajesh Balan, Amit Manjhi, Yiannis Koutis, Stavros Harizopoulos, Stratos Papadomanolakis, Bianca Schröder, Tiankai Tu, Julio Lopez.

The Computer Science Department and Carnegie Mellon has been an excellent educational environment which supported me throughout the course of this work. I'd like to thank all of my teachers. I would like to thank all those who made the gears run smoothly, including Sharon Burks, Deborah Cavlovich, Catherine Copetas, and Barbara Grandillo.

I would like to thank my feline friends Kiki and Aki.

Most of all, I am grateful to my family, Mom and Dad, my sister Heejoo and brother Kangwon, my best friend and now my husband Spiros Papadimitriou and his parents. Without their constant love, support, and encouragement, none of this would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Attacks by Internet worms, viruses, and unwanted e-mails such as spams have threatened the Internet since its inception [118]. Although much work has been done to prevent such attacks, the threat does not seem to diminish. Rather, the volume, intensity, and severity of attacks has increased as modern society relies more on the Internet, and attackers find the ways to profit from them [132, 143, 53]. A trend analysis reports that since July 2002 the number of known viruses has increased from 75,000 to more than 115,000 by December 2005 [103].

Attacks aim to reach as many computers and users as possible in a short period of time. Internet worms and viruses expedite their propagation by using infected computers to infect the next victims. Spammers launch million copies of advertisement messages to distributed hosts using automated spam tools from multiple computers. Such *wide-spreading malicious payloads* share some common communication patterns: 1) relying on many-to-many communication to reach multiple thousands of computers quickly, and 2) having unique byte patterns that are invariant across all the payload instances.[1] The byte patterns that distinguish wide-spreading malicious payloads from other legitimate traffic are called the *signatures* of the malicious payload.

Our thesis is that *content prevalence analysis in network payloads across distributed networks is a good basis for automated signature generation for wide-spreading malicious payloads, and can be performed without compromising the privacy of participating networks*.

In this dissertation, we show the validity of our thesis by applying the privacy-preserving distributed content prevalence analysis approach to develop automated signature genera-

[1] It is believed that spam e-mails have varying content to avoid spam filtering. However, many of spam e-mails still have advertisers' URLs or similar contact information the potential clients can use to contact them, and such information is hard to change.

tion systems for novel Internet worms. The approach is generally applicable to a broader range of wide-spreading malicious payload detection such as spam messages and network virus detection, if such malicious activities involve the transmission of unique, identical payload patterns from multiple computers to a large number of computers.

The rest of this chapter examines the traffic patterns of wide-spreading malicious payload transmission; gives an overview of automated worm containment based on content-based filtering; introduces our technical approach to build the automated signature generation system; and discusses the key contributions of this dissertation.

1.1 Wide-spreading Malicious Payloads

Modern Internet attacks attempt to infect or damage as many victims as possible before being detected. Thus, they often employ automated propagation techniques and launch attacks from multiple hosts. In this section, we survey three common Internet attacks: worms, e-mail spams, and blog comment spams. The survey suggests that the traffic from these attacks exhibits the following characteristics of typical wide-spreading malicious payloads:

- **Content Invariance:** all flows over which wide-spreading malicious payloads spread contain an identical, invariant portion of payload if it exploits a specific vulnerability or its goal is to advertise a certain piece of information.
- **Wide-spreading Behavior:** they propagate from many sources to many destinations so that they can contact millions of destinations quickly.

1.1.1 Network Viruses and Worms

Network viruses and worms are self-replicating programs. Originally, the term *virus* was used for a program that inserts a possibly evolved copy of itself into a host, including operating systems, to propagate. Unlike a virus that requires a user to activate it, a *worm* is self-contained and propagates without any human intervention by exploiting a software vulnerability on remote computers [113]. Elimination of human intervention allows a carefully crafted worm to spread over many vulnerable hosts at high speed. Infectious programs such as the Nimda worm [20] propagate using both virus-like and worm-like schemes so that the distinction between viruses and worms has blurred.

After the Morris worm infected 10% of ARPANET hosts [37], worms and network viruses have drawn much attention from the research community and the governments. However,

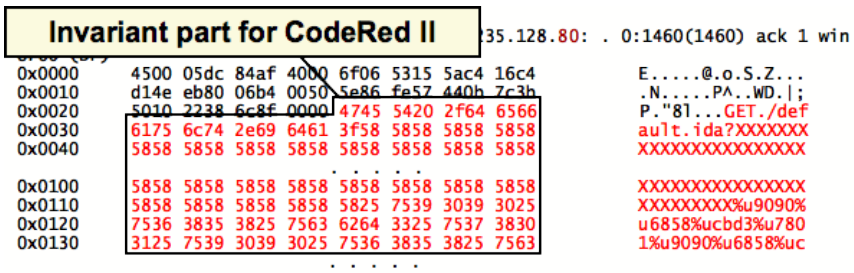


Figure 1.1: **An invariant payload pattern appeared in CodeRed II infection payloads:** the invariant part with the destination port number is used as the signature of CodeRedII worm by signature-based Intrusion Detection Systems.

the number of those viruses and worms has continued to grow relentlessly. According to the analysis by Sophos Plc. [103], since July 2002 the number of known viruses and worms has increased from 75,000 to more than 115,000 by December 2005. As shown by the Code-Red worm (CRv2) and the sequence of following worms, a worm can compromise most network-attached computers with a vulnerability in a very short time. The Code-Red worm infected 359,000 hosts in less than 14 hours [85] and the Slammer worm infected 90% of vulnerable hosts within 10 minutes.

Content Invariance: A worm remotely exploits a software vulnerability or a set of vulnerabilities on a victim host, such that the victim becomes infected, and itself begins remotely infecting other victims. That commonality in functionality has to date led to commonality in code, and thus in payload content, across worm infection payloads transmitted by the worm. Such an invariant part in worm payloads is used as a signature of the worm that distinguishes the worm traffic from other traffic. For instance, Figure 1.1 shows the invariant part in all the CodeRed II worm payloads. The part triggered a buffer overflow in Microsoft's Internet Information Server (IIS) and switched the program control to an arbitrary code included in a later part of the payload. This invariant payload string served as the signature of the CodeRed II until all the vulnerable IIS servers in the Internet were patched.

Wide-spreading Behavior: Once infected, the newly infected hosts participate in transmitting infection payloads, which results in the invariant payload patterns transferred from many hosts to many hosts.

1.1.2 E-mail Spams

Although the focus of this thesis is on worms, it is interesting to note that different types of spams share the same patterns of content invariance and wide-spreading behavior. For

example, e-mail spam has become one of the greatest threats to the use of e-mail as a form of communication. The Messaging Anti-Abuse Working Group (MAAWG) estimates that about 80% of incoming mail is "abusive email"[2], as of the 1st quarter of 2006, after analyzing 435.6 million mailboxes [83].

The severity of spams goes far beyond mere inconvenience. Unsolicited e-mail spams cost average U.S. organizations 1.4% of employee productivity, or $874 per employee per year in 2003 [98]. Note that the total cost would be much higher if we included the extra overhead due to overloaded e-mail systems and missing important e-mails. Besides advertisement, e-mail spams are also used for more malicious purposes, such as scam distribution, personal information collection, and virus propagation.

Content Invariance: Many spam-filtering techniques work by searching for patterns in headers or bodies of e-mail messages. In order to avoid being detected, e-mail spams obfuscate their message content by intentionally misspelling commonly-filtered words, inserting HTML comments between letters, or presenting the text as an image. Note, however, that the common purpose of spamming is to advertise products or to attract people to a fake web site. Thus, spams often include URLs, phone numbers, or mail addresses. Obfuscating such contact information is not impossible but requires more effort than changing other parts of e-mail messages. Our analysis with traces from a spam honeypot (total 411K spam e-mails during a week of Jan 2006) showed that 55% of e-mail spams contain some contact information in the forms of URL or e-mail addresses, even though content obfuscation is widely used in the spamming community.

Wide-spreading Behavior: Spam traffic exhibits the many-to-many communication pattern. By definition, spams are sent to a large number of users. Because almost all Internet service providers and mail service providers forbid the use of their services to send spam, spammers make use of open relays/proxies, bots, and infected hosts distributed in the Internet. Utilizing the fact that many e-mail spams hold the two properties of wide-spreading malicious payloads, several spam filtering techniques [133] have been proposed to detect spams.

1.1.3 Link Spams

Since mid-1990's, link-based ranking algorithms, such as Google's PageRank algorithm [13] have been widely used for web search. The rise of link-based ranking algorithms has spawned

[2]Because the precise definition of spam differs slightly from jurisdiction to jurisdiction in accordance with local laws, MAAWG avoids using the term "spam" in this report. Abusive emails are communications that seek to exploit the end user.

a new type of spams called *link spam*.

Link spam is distributed by automatically posting random comments to publicly accessible web applications that accept and display hyperlinks submitted by visitors. Blogs, public bulletin boards, web guestbooks, forums, and wikis are good targets for link spams. Adding hyperlinks (URL) pointing to the spammers' intended web sites artificially increases the sites' ranking in web search results computed with link-based ranking algorithms. An increased ranking often results in the spammer's commercial site being listed ahead of other sites, increasing the number of potential visitors and customers.

Even though most web applications attempt to provide various filtering methods to prevent or reduce such annoying link spams, spammers have become smarter. Akismet, a blog spam filtering service [2], reports that spam comments account for 95% out of all comments reported to the service, and the number of link spam continues to grow.

Content Invariance: Unlike e-mail spams, link spams always contain URL information pointing to the intended web sites. It is not impossible for spammers to create link farms [56], with multiple web pages and make those pages point to an intended web site. However, utilizing many host and domain names is not as simple as e-mail spam message obfuscation. Spammers, thus, often reuse the same URLs or domain names when they post link spam messages in open blogs and open bulletin board systems. Those URLs posted by spammers can be used as a signature to distinguish spam messages from legitimate messages.

Wide-spreading Behavior: Like other types of spam, link spam aims to be distributed in many web pages. In order to spread spam links on many web pages efficiently, spammers use automated spamming tools running on multiple computers, possibly a group of compromised machines or bots under attackers' control.

Recently, these two properties have inspired collaborative spam filtering services such as LinkSleeve [79], Akismet [2], and Eolin Anti-Spam [130] services. These techniques strip posted message of included URLs and keywords, and count the number of blogs that received the same URLs. If a threshold is exceeded, the message is more likely to be a spam.

1.2 Worm Defense and Challenges

Internet worms have exploited the relative lack of diversity in the software on Internet-attached hosts, and the ease with which those hosts can communicate. A worm program is self-replicating. It remotely exploits a software vulnerability on a victim host, such that the victim becomes infected, and itself begins remotely infecting other victims. The severity of the worm threat goes far beyond mere inconvenience. The total cost of the Code Red worm

epidemic, as measured in lost productivity owing to interruptions in computer and network services, is estimated at $2.6 billion. Technology market researcher Computer Economics estimated that the SQL Slammer worm cost between $750 million and $1 billion to clean up, which seemed less costly than the Code Red worm epidemic that impacted more machines. However, considering that the Slammer worm clogged and downed financial systems such as banks networks and ATMs, many security experts argued that the Slammer worm was worse than the Code Red worm [75]. A worm can install backdoors and abuse infected hosts for Denial-of-Service attacks, spamming, and private information leakage. For example, many worms such as the Sasser worm [33], open backdoors on infected hosts and form a large botnet [107] through which worm writers can control compromised hosts. Worm writers may choose to sell access to infected machines to black hats, which creates financial incentives for writing malicious worms [114].

Because worms can spread too rapidly for humans to respond [127, 140], automation of worm defense is a key element in preventing worms. In recent years, the research community has proposed various methods for automatically *detecting* and *containing* worms by quarantining infected hosts or filtering worm traffic. Broadly speaking, three chief strategies exist to filter worm traffic: 1) blocking all traffic destined for the ports on which worms spread (*port blocking*), 2) blocking all traffic for the relevent ports from worm-infected hosts (*source-address blocking*), and 3) blocking all flows that exhibit anomalous payload or communication patterns (signatures) that only a worm shows in its infection attempts (*signature-based filtering*). Filtering all traffic based on destination port alone can block legitimate Internet traffic. Moore *et al.* [89] compared the relative efficacy of source-address filtering and signature-based filtering. Their results show that signature-based filtering impedes the spreading of a worm more effectively, provided the signatures are discovered and deployed before the worm starts its exponential growth.

We can classify the signature into two groups based on what aspects of worm's property it encodes:

- **Network payload signatures** are the unique byte patterns occurring in worm's network payloads. Signature-based filters that use network payload signatures (*content-based filtering*) can block worm traffic within both networks and transport layers of hosts. They do not need application-specific knowledge more than payload patterns. Cisco's nbar [28], bro [102], snort [131] are examples of content-based filters using network payload signatures. Network payload signatures are simple and efficient to use. These content-based filtering techniques are quite mature compared to techniques using different types of signatures. However, it is sometimes impossible to define net-

work payload signatures if a worm uses strong polymorphism or propagates over encrypted channels. In this thesis, we focus on the systems that generate network payload signatures.

- **Host payload signatures** used in Shield [136], Vigilante [34], are usable by filters installed immediately above the network stack of a host. Application-specific knowledge is required to generate those signatures. Because the filters operate at the application layer, they can handle IPSec-encrypted traffic. Encrypted messages above the transport layer are still difficult to detect, but researchers envision that appropriate programmable hooks, provided by applications and libraries, can enable to define more powerful signatures and filtering policy [136]. Also, more powerful comparisons thansimple pattern matching is feasible because the volume of traffic one host deals with would be much less than the aggregated volume of traffic a content-based filter on network links has to handle. For example, one can choose a signature that tests if the length of a GET request copied to the stack exceeds 240 bytes in order to detect worms that exploit IIS `.ida` vulnerability.

- **Network behavioral signatures** [43] describe anomalous, spatial and temporal communication patterns observed from networks during a worm infection process. For example, a host infected by an aggressive, random-scanning worm may contact a significant number of hosts, and we can encode this fact as a network behavioral signature. A worm that recruit participants of a botnet would make a connection to one of the master servers that control the botnet. Then, the list of master servers would be a signature if they are not used for legitimate communication. Behavioral signatures can catch polymorphic and encrypted worms if the worms show obviously abnormal communication patterns. If normal traffic pattern changes or the worms' propagation patterns are similar to a normal traffic pattern, the signatures can cause false positives. Compared to filters using payload signatures, filters using network behavioral signatures require more memory and computation power because they must maintain all the communication history useful for correlation. Some systems such as Bro [102] can utilize behavioral signatures (policy, in bro language) but an automated system to generate behavioral signatures is not yet available.

The basic premise when using signature-based filtering for automatic worm containment is that signatures of a worm must be known beforehand. Unfortunately, today's signature detection process requires human involvement, and thus is slow. For instance, when Ben

Koshy[3] from W3 International Media first reported the Slammer worm propagation incident to a mailing list [4], more than 5 hours had already passed since the worm was released. The delay included the time to notice a worm outbreak via out-of-band communication (i.e., emails, newsgroups, phone, security companies' help desks), capture worm traces, select signatures based on manual analysis, and publish/deploy them. According to Moore *et al.* [89], signatures should be deployed within 60 minutes of a worm's release to contain slow-spreading worms such as CodeRed, and within less than 5 minutes to contain fast-spreading worms such as Slammer. Staniford *et al.* citeWeaverWarchol demonstrated that a worm that is carefully designed to efficiently scan victims (hitlist, permutation-scanning, topological scanning, and efficient threading) can infect most vulnerable hosts before any human intervention. Automation of worm signature detection is required to achieve effective automated worm containment with content-based filtering.

An automated signature generation system must 1) generate signatures that catch *all* possible variants of a worm, but not block legitimate traffic and 2) generate signatures quickly enough to prevent global infection, 3) require minimal real-time operator intervention, 4) minimize the effect on legitimate Internet use, and 5) preserve privacy if information needs to be exchanged among multiple monitoring points.

In this dissertation, we present the design and implementation of a distributed, automated worm signature generation system, *Autograph*, and evaluate the system based on the aforementioned requirements in designing an automated signature generation system. Autograph combines information from distributed monitors within an enterprise network or in honeypots [125]. For fast and accurate signature generation, monitors from different sites in the Internet can also exchange their observations. The signatures determined by Autograph can then be disseminated automatically to content filters in networks and hosts before the worm infects a significant fraction of vulnerable hosts.

1.3 Technical Approaches

As discussed in Section 1.1.1, worms have invariant parts in their payloads and their propagation shows a typical many-to-many communication pattern. We solve the automated signature generation problem by exploiting these properties:

[3]Ben Koshy has been credited with being one of the first persons to identify the Slammer worm.

[4]Archives of the NTBUGTRAQ mailing list. `http://listserv.ntbugtraq.com/archives/ntbugtraq.html`

- **Content-prevalence analysis**: Autograph analyzes the content of network flow payloads and chooses the most frequently occuring byte patterns as signatures. Two properties of worms suggest that content-prevalence analysis is fruitful. First, all flows over which a worm spreads contain an identical, invariant portion of payload if the worm exploits a specific vulnerability. Second, a worm generates voluminous network traffic as it spreads, because of its self-propagating nature.

- **Distributed monitoring**: Autograph takes advantage of distributed monitoring. The volume of worm traffic one monitor observes will be small at the early stage of a worm epidemic or for slowly propagating worms. However, the aggregated volume that multiple monitors receive will be large. Intuitively, the more monitors we deploy, the more quickly we capture worm traffic. When a worm does not propagate randomly or the vulnerability density across the Internet is not uniform, distributed monitoring helps shorten the time needed to catch worm traffic. Moreover, distributed monitoring allows Autograph to utilize many exploit detectors with different server versions and configurations, to verify the quality of generated signatures.

- **Privacy-preserving payload sharing**: Autograph expedites signature detection by aggregating suspicious flow payloads observed by distributed monitors. However, this payload sharing may compromise privacy. To allow privacy-protecting payload sharing, we rely on the observation that a single payload observed at many sites should not be private, and thus may be shared as long as the payload providers are not disclosed.

1.4 Contributions and Dissertation Outline

This dissertation explores approaches for automatically and quickly generating Internet worms. Here are the specific contributions:

1. **Distributed, automated worm signature generation**: We present a new system, called *Autograph*, that automatically generates payload signatures for Internet worms by combining suspicious flow selection heuristics and content-prevalence analysis on selected flows (Chapter 3). We extend the system to share port scan reports among distributed monitor instances. Using trace-driven simulation, we demonstrate the value of this technique in speeding the generation of signature for novel worms (Chapter 4).

2. **Privacy-preserving distributed, content-prevalence analysis**: We present a distributed payload sharing framework that improves automated signature detection speed, and

validate our assumption on the privacy of globally prevalent payload pattern strings by examining real network traces (Chapter 6). In distributed payload sharing, privacy is the major concern. We present two privacy-preserving, distributed content-prevalence analysis techniques that disclose a payload pattern only when it belongs to worm traffic that is observed at multiple monitoring sites.

- The first technique meets the goal by applying **HotItemID** protocol [69], which relies an approximate counting techniques. This technique is suitable for an environment where a large number of autonomous sites or hosts participate, and when a worm attacks a large fraction of those participating sites.
- The second technique derives from a **privacy-preserving multiset operation** framework that relies on a semantically secure homomorphic cryptosystem and arithmetic operations over polynomial representation of sets [70]. Combined with sampling, this technique can scale from tens to hundreds of participating monitors. To our knowledge, this is the first work that applies Threshold-Set-Intersection framework in practice and evaluates its performance.

This thesis consists of two parts.

In Part I, we focus on the automated signature detection technique that exploits worm's unique properties. Chapter 2 discusses the desired properties of automated signature generation systems. Chapter 3 presents our system *Autograph* that automates signature generation process by content prevalence analysis. Chapter 4 extends Autograph to use distributed monitors for port-scanner detection.

In Part II, we explore techniques that allow information sharing beyond port-scanner IP addresses, so allows collaborative suspicious flow accumulation. The information sharing must preserve the privacy of participating networks. Chapter 5 motivates distributed payload sharing and discusses the privacy-related issues. Chapter 6 presents our distributed payload sharing framework and its signature detection speed. Chapter 7 explores the idea of hiding private data in a crowd using our HOTITEM-ID protocol. Chapter 8 presents another technique to build a privacy-preserving distributed signature generation system. The system uses the theoretically proven privacy-preserving Threshold-Set-Intersection framework.

Finally, we review the related work in Chapter 9 and conclude by summarizing our findings and contributions in Chapter 10.

Part I

Content-based Automated Signature Generation

Chapter 2

Introduction

Today's Internet intrusion detection systems (IDSes) monitor edge networks' DMZs [1] to identify and/or filter malicious flows. While an IDS helps protect the hosts on its local edge network from compromise and denial of service, it cannot alone effectively intervene to halt and reverse the spreading of novel Internet worms. Generation of the *worm signatures* required by an IDS—the byte patterns sought in monitored traffic to identify worms—today entails non-trivial human labor, and thus significant delay: as network operators detect anomalous behavior, they communicate with one another and manually study packet traces to produce a worm signature. Yet intervention must occur early in an epidemic to halt a worm's spread. In this part, we introduce Autograph, a system that *automatically* generates signatures for novel Internet worms that propagate by randomly scanning the IP address space. Autograph generates signatures by analyzing the *prevalence of portions of flow payloads*, and thus uses no knowledge of protocol semantics above the transport level. It is designed to produce signatures that exhibit high *sensitivity* (high true positives) and high *specificity* (low false positives); our evaluation of the system on real DMZ traces validates that it achieves these goals. We extend Autograph to share port scan reports among distributed monitor instances, and using trace-driven simulation, demonstrate the value of this technique in speeding the generation of signatures for novel worms. Our results elucidate the fundamental trade-off between early generation of signatures for novel worms and the specificity of these generated signatures.

[1] DMZ is a physical or logical subnetwork placed in between an organization's internal network and a larger, untrusted network such as the Internet. DMZ may contain and expose the organization's external services. External attackers only have access to the services and equipments in the DMZ, rather than the organization's internal network.

2.1 Background and Motivation

Motivated in large part by the costs of Internet worm epidemics, the research community has investigated worm propagation and how to thwart it. Initial investigations focused on case studies of the spreading of successful worms [85], and on comparatively modeling diverse propagation strategies future worms might use [127, 140]. More recently, attention has turned to methods for *containing* the spread of a worm. Broadly speaking, three chief strategies exist for containing worms by blocking their connections to potential victims: discovering ports on which worms appear to be spreading, and filtering all traffic destined for those ports; discovering source addresses of all traffic destined for those ports; discovering source addresses of infected hosts and filtering all traffic (or perhaps traffic destined for a few ports) from those source addresses; and discovering the payload content string that a worm uses in its infection attempts, and filtering all flows whose payloads contain that content string.

Detecting that a worm appears to be active on a particular port [145] is a useful first step toward containment, but is often too blunt an instrument to be used alone; simply blocking all traffic for port 80 at edge networks across the Internet shuts down the entire web when a worm that targets web servers is released. Moore *et al.* [89] compared the relative efficacy of source-address filtering and content-based filtering. Their results show that content-based filtering of infection attempts slows the spreading of a worm more effectively: to confine an epidemic within a particular target fraction of the vulnerable host population, one may begin content-based filtering far later after the release of a worm than address-based filtering. Motivated by the efficacy of content-based filtering, we seek in this thesis to answer the complementary question unanswered in prior work: *how should one obtain worm content signatures for use in content-based filtering?*

Here, a *signature* is a tuple `(IP-proto, dst-port, byteseq)`, where `IP-proto` is an IP protocol number, `dst-port` is a destination port number for that protocol, and `byteseq` is a variable-length, fixed sequence of bytes.[2] Content-based filtering consists of matching network flows (possibly requiring flow reassembly) against signatures; a match occurs when `byteseq` is found within the payload of a flow using the `IP-proto` protocol destined for `dst-port`. We restrict our investigation to worms that propagate over TCP in this work, and thus hereafter consider signatures as `(dst-port, byteseq)` tuples.

Today, there exist TCP-flow-matching systems that are "consumers" of these sorts of sig-

[2]Signatures may employ more complicated payload patterns, such as regular expressions. We restrict our attention to fixed byte sequences.

natures. Intrusion detection systems (IDSes), such as Bro [3] and Snort [4], monitor all incoming traffic at an edge network's DMZ, perform TCP flow reassembly, and search for known worm signatures. These systems log the occurrence of inbound worm connections they observe, and can be configured (in the case of Bro) to change access control lists in the edge network's router(s) to block traffic from source IP addresses that have sent known worm payloads. Cisco's NBAR system [5] for routers searches for signatures in flow payloads, and blocks flows on the fly whose payloads are found to contain known worm signatures. We limit the scope of our inquiry to the *detection and generation* of signatures for use by these and future content-based filtering systems.

It is important to note that all the content-based filtering systems use databases of worm signatures that are *manually* generated: as network operators detect anomalous behavior, they communicate with one another, manually study packet traces to produce a worm signature, and publish that signature so that it may be added to IDS systems' signature databases. This labor-intensive, human-mediated process of signature generation is slow (on the order of hours or longer), and renders today's IDSes unhelpful in stemming worm epidemics—by the time a signature has been found manually by network operators, a worm may already have compromised a significant fraction of vulnerable hosts on the Internet.

We seek to build a system that automatically, without foreknowledge of a worm's payload or time of introduction, detects the signature of any worm that propagates by randomly scanning IP addresses. We assume the system monitors all inbound network traffic at an edge network's DMZ. *Autograph,* our worm signature detection system, has been designed to meet that goal. The system consists of three interconnected modules: a flow classifier, a content-based signature generator, and *tattler*, a protocol through which multiple distributed Autograph monitors may share information, in the interest of speeding detection of a signature that matches a newly released worm.

2.2 Desirable Properties

While the initial motivation of automated signature detection is to speed the signature generation by taking the human out of signature generation process, we must also ensure that the system generates good signatures and consume reasonable amount of resources. More

[3] Bro Intrusion Detection System [102]. `http://www.bro-ids.org`

[4] The Snort Project. `http://www.snort.org/`

[5] More information is at `http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122newft/122t/122t8/dtnbarad.htm`

		True Positive	
		High	Low
False Positive	High	Sensitive, Unspecific	Insensitive, Unspecific
	Low	Sensitive, Specific	Insensitive, Specific

Figure 2.1: **Combinations of sensitivity and specificity.**

specifically, developing and evaluating an automated signature detection system must consider the following metrics:

- **Signature Quality:** Ideally, the quality of the signatures generated with an automated signature detection system should be comparable to or even better than that of the signatures human experts generate. In describing the efficacy of worm signatures in filtering worm traffic, we adopt the parlance used in epidemiology to evaluate a diagnostic test:

  - *Sensitivity* relates to the *true positives* generated by a signature; among all the worm flows crossing the network, the fraction of the worm flows matched, and thus successfully identified, by the signature. Sensitivity is reported as $t \in [0, 1]$, the fraction of true positives among worm flows. Note that another frequently used metric, false negative rate is $1 - t$.
  - *Specificity* relates to the *false positives* generated by a signature; in a mixed population, the fraction of non-worm flows matched by the signature, and thus incorrectly identified as worms. Specificity is typically reported as $(1 - f) \in [0, 1]$, where f is the fraction of false positives among non-worm flows.

  Throughout this part, we classify signatures according to this terminology, as shown in Figure 2.1. A good signature system generates sensitive and specific signatures.

  In practice, there is a tension between sensitivity and specificity; one often suffers when the other improves, because a diagnostic test (*e.g.,* "is this flow a worm or not?") typically measures only a narrow set of features in its input, and thus does not perfectly classify it. There may be cases where inputs present with identical features in the eyes of a test, but belong in different classes.

- **Signature quantity and length:** A content-based filter maintains a list of signatures and matches every packet (or data injected from networks) against the signatures known for the IP protocol and port. In practice, network administrators maintain the list as small as possible because the filter's performance degrades linearly with the number of signatures in the list [5]. For example, Snort [131] users use tens of signatures only for active, serious attacks, even though there are more than 3000 rules available currently. Thus, fewer signatures that cover a wide range of worms are desirable.

  Similarly, the cost of signature matching increases with the length of each signature. For example, when we express the signature with a regular expression which is implemented as a Deterministic Finite Automation (DFA), the matching always takes time linear in the length of the signature regardless of the specifics of the signature. The parallel Boyer-Moore approaches have been explored for fast signature matching of multiple fixed strings [30, 49], and they have been adopted in Snort. However, they present a wide range of running times – potentially sublinear but also potentially superlinear in the length of the string. A different approach using Bloom filters in Field Programmable Gate Arrays (FPGA) has been proposed to speed network packet payload pattern matching [38]. This technique requires to group and store the signatures in parallel Bloom filters based on their lengths, so limiting the lengths of signatures is still important for its performance. Therefore, short signatures are preferable to long ones. Note that the signature length profoundly affects specificity and sensitivity: when one signature is a subsequence of another, the longer one is expected to match fewer flows than the shorter one.

- **Timeliness of detection:** Left unchecked by patches, traffic filtering, or other means, worms infect vulnerable hosts at an exponential rate, until the infected population saturates. Provos [105] shows in simulation that patching of infected hosts is more effective the earlier it is begun after the initial release of a new worm, and that in practical deployment scenarios, patching must begin quickly (before 5% of vulnerable hosts become infected) in order to have hope of stemming an epidemic such that no more than 50% of vulnerable hosts ever become infected. Moore *et al.* [89] show similarly that signature-based filtering of worm traffic stops worm propagation most effectively when begun early.

- **Automation:** This metric is tightly related to the timeliness of detection. A signature detection system must require minimal real-time operator intervention. Vetting sig-

natures for specificity with human eyes, *e.g.,* is at odds with timeliness of signature detection for novel worms.

- **Efficient resource consumption:** A signature detection system that monitors host state must not affect host performance significantly, so CPU and memory usage must be small. A signature detection system that monitors network traffic must keep up with the large volume of aggregated traffic on fast links, so signature detection algorithms must be simple and fast. If a signature detection system is deployed in a distributed fashion, such that distributed monitors communicate with one another about their observations, that communication should remain scalable, even when a worm generates tremendous network activity as it tries to spread. That is, monitor-to-monitor communication should grow slowly as worm activity increases.

- **Robustness against attacks and polymorphism:** A *polymorphic* worm changes its payload in successive infection attempts. Such worms pose a particular challenge to match with signatures, as a signature sensitive to a portion of one worm payload may not be sensitive to any part of another worm payload. If a worm were "ideally" polymorphic, each of its payloads would contain no byte sequence in common with any other, and content-based filtering is no longer effective for such worms. However, if a worm exhibits polymorphism but does not change one or more relatively long subsequences across its variants, an efficient signature detection system will generate signatures that match these invariant subsequences. This will minimize the number of signatures required to mach all the worm's variants. Also, a signature detection system must be robust against abuse, such that an attacker forces the system to generate unspecific signatures. Those signatures will result in Denial-of-Service attacks against networks using automatically generated signatures.

2.3 Outline of Part I

In the remainder of Part I, we present an automated signature generation system that satisfies the above requirements. In Chapter 3, we design Autograph, our worm signature detection system that analyzes traffic of a *single* network. In Chapter 4, we extend Autograph in a *distributed* monitoring setting to enable fast signature generation.

Chapter 3

Automated Worm Signature Generation using Content Prevalence Analysis

In this chapter we propose a new system, *Autograph*, that generates worm signatures by detecting prevalent payload pattern strings across suspicious traffic flows. The main contributions of our work can be summarized as follows:

- We introduce a novel *pattern-extraction-based* network signature generation approach based on the two properties of worms: *content invariance* and magnitude of traffic volume. These two traits of worm traffic were exploited by many other pattern-extraction-based network signature generation systems proposed concomitantly [119, 73] and after our work [93, 139, 67, 17, 76, 94].

- We propose COPP, a payload partitioning algorithm derived from a work in the file system domain [91]. COPP generates variable-size, non-overlapping content blocks by determining the boundaries of each block based on payload content. COPP is resilient against modest level of payload pattern changes.

- We show that *pattern-extraction-based* network signature generation approaches generally suffer from the advent of new Internet applications such as P2P file sharing programs because their communication patterns are similar to that of worm traffic. In order to avoid false positives caused by such benign P2P application traffic, Autograph needs to apply better flow selection heuristics.

We provide the overview of a single Autograph monitor in Section 3.1. We then describe the detailed workings of Autograph's suspicious flow selection classifier and content-based

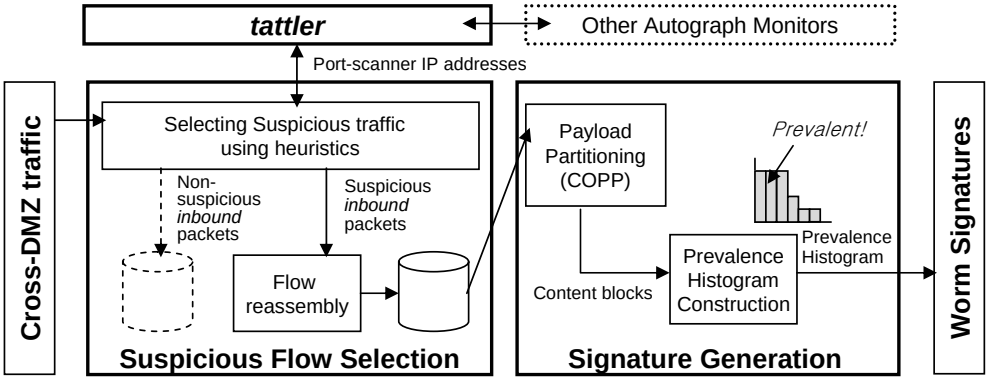


Figure 3.1: **Architecture of an Autograph Monitor:** A single Autograph monitor consists of 1) a suspicious flow selection stage and, 2) a content-based signature generation stage.

signature generation in Section 3.2 and Section 3.3 respectively. We evaluate the quality of the signatures Autograph finds when run on real DMZ traces from two edge networks in Section 3.4. In Section 3.5, we study the P2P file sharing application traffic that mimics worm traffic and explain how Autograph can cope with the noise caused by the new application. After cataloguing limitations of Autograph and possible attacks against it in Section 3.6, we summarize our work in Section 3.7.

3.1 Autograph System Overview

Motivated by the design goals given in the previous chapter, we now present Autograph. We begin with a schematic overview of the system, shown in Figure 3.1. A single Autograph monitor's input is all traffic crossing an edge network's DMZ, and its output is a list of worm signatures. We defer discussion of tattler, used in distributed deployments of Autograph, to Chapter 4. There are two main stages in a single Autograph monitor's analysis of traffic. First, a *suspicious flow selection* stage uses heuristics to classify inbound TCP flows as either suspicious or non-suspicious.

After classification, packets for these inbound flows are stored on disk in a *suspicious flow pool* and *non-suspicious flow pool*, respectively. For clarity, throughout this thesis, we refer to the output of the classifier using those terms, and refer to the *true* nature of a flow as *malicious* or *innocuous*. Further processing occurs *only* on payloads in the suspicious flow pool. Thus, flow classification reduces the volume of traffic that must be processed subsequently. We assume in our work that such heuristics will be far from perfectly accurate. Yet any heuristic that generates a suspicious flow pool in which truly malicious flows are a greater fraction of

flows than in the total inbound traffic mix crossing the DMZ will likely reduce generation of signatures that cause false positives, by focusing Autograph's further processing on a flow population containing a lesser fraction of innocuous traffic. Autograph performs TCP flow reassembly for inbound payloads in the suspicious flow pool. The resulting reassembled payloads are analyzed in Autograph's second stage, *signature generation*.

We stress that Autograph segregates flows by destination port for signature generation; in the remainder of this thesis, one should envision one separate instance of signature generation for each destination port, operating on flows in the suspicious flow pool destined for that port. Signature generation involves analysis of the *content* of payloads of suspicious flows to select sensitive and specific signatures. Two properties of worms suggest that content analysis may be fruitful. First, a worm propagates by exploiting one software vulnerability or a set of such vulnerabilities. That commonality in functionality has to date led to commonality in code, and thus in payload content, across worm infection payloads. In fact, Internet worms to date have had a single, unchanging payload in most cases. Even in those cases where multiple variants of a worm's payload have existed (*e.g.,* Nimda), those variants have shared significant overlapping content.[1] Second, a worm generates voluminous network traffic as it spreads; this trait stems from worms' self-propagating nature. For port-scanning worms, the exponential growth in the population of infected hosts and attendant exponential growth in infection attempt traffic are well known [85]. As also noted and exploited by Singh *et al.* [119], taken together, these two traits of worm traffic—content commonality and magnitude of traffic volume—suggest that analyzing the frequency of payload content should be useful in identifying worm payloads. During signature generation, Autograph measures the frequency with which non-overlapping payload substrings occur across all suspicious flow payloads, and proposes the most frequently occurring substrings as candidate signatures.

3.2 Selecting Suspicious Traffic

In this work, we use a simple port-scanner detection technique as a heuristic to identify malicious traffic; we classify all flows from port-scanning sources as suspicious. Note that we do not focus on the design of suspicious flow classifiers herein; Autograph can adopt *any* anomaly detection technique that classifies worm flows as suspicious with high probability. In fact, we deliberately use a port-scanning flow classifier because it is simple, computa-

[1] Future worms may be designed to minimize the overlap in their successive infection payloads; we consider such worms in Section 3.4.2.

tionally efficient, and *clearly imperfect*; our aim is to demonstrate that Autograph generates highly selective and specific signatures, even with a naive flow classifier. With more accurate flow classifiers, one will only expect the quality of Autograph's signatures to improve.

Many worms rely on scanning of the IP address space to search for vulnerable hosts while spreading. If a worm finds another machine that runs the desired service on the target port, it sends its infectious payload. Probing a non-existent host or service, however, results in an unsuccessful connection attempt, easily detectable by monitoring outbound ICMP host/port unreachable messages, or identifying unanswered inbound SYN packets. Hit-list worms [127, 88] violate this port-scanning assumption; we do not address them in this thesis, but comment on them briefly in Section 3.6.

Autograph stores the source and destination addresses of each inbound unsuccessful TCP connection it observes. Once an external host has made unsuccessful connection attempts to more than s internal IP addresses, the flow classifier considers it to be a scanner. All successful connections from an IP address flagged as a scanner are classified as suspicious, and their inbound packets written to the suspicious flow pool, until that IP address is removed after a timeout (24 hours in the current prototype).[2] Packets held in the suspicious flow pool are dropped from storage after a configurable interval t. Thus, the suspicious flow pool contains all packets received from suspicious sources in the past time period t.[3]

Autograph reassembles all TCP flows in the suspicious flow pool, continuously examines the suspicious flow pool that contains all the reassembled TCP flows for the last t minutes. The signature generation process is initiated when for a single destination port, the suspicious flow pool contains more than a threshold number of flows θ.

Honeypots [125] are useful sources of suspicious traffic for Autograph. Because all traffic arriving at a honeypot is inherently suspicious, no port scan detection may be necessary; all payloads received at a honeypot may be included in Autograph's suspicious traffic pool. However, remember that the incoming traffic to Honeypot still can contain innocuous contents; for instance, an attacker can send a common connection request to determine whether the destination is a honeypot or a valid server. Thus, we still need the content-based analysis to extract sensitive and selective signatures.

[2]Note that an IP address may have sent traffic before being identified as a scanner; such traffic will stored in the non-suspicious flow pool. We include only *subsequently* arriving traffic in the suspicious flow pool, in the interest of simplicity, at the expense of potentially missing worm traffic sent by the scanner before our having detected it as such.

[3]Worms that propagate very slowly may only accumulate in sufficient volume to be detected by Autograph for long values of t.

3.3 Content-Based Signature Generation

After selecting suspicious flows, Autograph selects the most frequently occurring byte sequences across the flows in the suspicious flow pool as signatures. To do so, it divides each suspicious flow into smaller content blocks, and counts the number of suspicious flows in which each content block occurs. We term this count a content block's *prevalence*, and rank content blocks from most to least prevalent. As previously described, the intuition behind this ranking is that a worm's payload appears increasingly frequently as that worm spreads. When all worm flows contain a common, worm-specific byte sequence, that byte sequence will be observed in many suspicious flows, and so will be highly ranked.

3.3.1 COPP: COntent-based Payload Partitioning Algorithm

Let us first describe how Autograph divides suspicious flows' payloads into shorter blocks. One might naively divide payloads into fixed-size, non-overlapping blocks, and compute the prevalence of those blocks across all suspicious flows. That approach, however, is brittle if worms even trivially obfuscate their payloads by reordering them, or inserting or deleting a few bytes. To see why, consider what occurs when a single byte is deleted or inserted from a worm's payload; all fixed-size blocks beyond the insertion or deletion will most likely change in content. Thus, a worm author could evade accurate counting of its substrings by trivial changes in its payload, if fixed-size, non-overlapping blocks were used to partition payloads for counting substring prevalence.

Instead, as first done in the file system domain in LBFS [91], we divide a flow's payload into *variable-length* content blocks using COntent-based Payload Partitioning (COPP). Because COPP determines the boundaries of each block based on payload content, the set of blocks COPP generates changes little under byte insertion or deletion.

To partition a flow's payload into content blocks, COPP computes a series of Rabin fingerprints r_i over a sliding k-byte window of the flow's payload, beginning with the first k bytes in the payload, and sliding one byte at a time toward the end of the payload. It is efficient to compute a Rabin fingerprint over a sliding window [108]. As COPP slides its window along the payload, it ends a content block when r_i matches a predetermined *breakmark*, B; when $r_i \equiv B \pmod{a}$. The average content block size produced by COPP, a, is configurable; assuming random payload content, the window at any byte position within the payload equals the breakmark $B \pmod{a}$ with probability $1/a$.

Figure 3.2 presents an example of COPP, using a 2-byte window, for two flows f_0 and f_1.

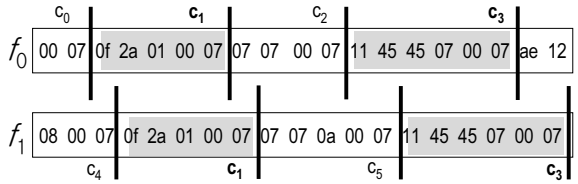


Figure 3.2: **COPP with a breakmark of** $Rabinfingerprint(\textbf{“0007”})$

Sliding a 2-byte window from the first 2 bytes to the last byte, COPP ends a content block c_i whenever it sees the breakmark equal to the Rabin fingerprint for the byte string “0007”. Even if there exist byte insertions, deletions, or replacements between the two flows, COPP finds identical c_1 and c_3 blocks in both of them.

Because COPP decides content block boundaries probabilistically, there may be cases where COPP generates very short content blocks, or takes an entire flow's payload as a single content block. Very short content blocks are highly unspecific; they will generate many false positives. Taking the whole payload is not desirable either, because long signatures are not robust in matching worms that might vary their payloads. Thus, we impose minimum and maximum content block sizes, m and M, respectively. When COPP reaches the end of a content block and fewer than m bytes remain in the flow thereafter, it generates a content block that contains the last m bytes of the flow's payload. In this way, COPP avoids generating too short a content block, and avoids ignoring the end of the payload.

3.3.2 Selecting Prevalent Content Blocks

After Autograph divides every flow in the suspicious flow pool into content blocks using COPP, it discards content blocks that appear only in flows that originate from a single source IP address from further consideration. We found early on when applying Autograph to DMZ traces that such content blocks typically correspond to misconfigured or otherwise malfunctioning sources that are *not malicious*; such content blocks typically occur in many innocuous flows, and thus often lead to signatures that cause false positives. Singh *et al.* [119] also had this insight—they consider flow endpoint address distributions when generating worm signatures.

Suppose there are N distinct flows in the suspicious flow pool. Each remaining content block matches some portion of these N flows. Autograph repeatedly selects content blocks as signatures, until the selected set of signatures matches a configurable fraction w of the flows in the suspicious flow pool. That is, Autograph selects a signature set that “covers” at

least wN flows in the suspicious flow pool.

We now describe how Autograph greedily selects content blocks as signatures from the set of remaining content blocks. Initially the suspicious flow pool F contains all suspicious flows, and the set of content blocks C contains all content blocks produced by COPP that were found in flows originating from more than one source IP address. Autograph measures the prevalence of each content block—the number of suspicious flows in F in which each content block in C appears—and sorts the content blocks from greatest to least prevalence. The content block with the greatest prevalence is chosen as the next signature. It is removed from the set of remaining content blocks C, and the flows it matches are removed from the suspicious flow pool, F. This entire process then repeats; the prevalence of content blocks in C in flows in F is computed, the most prevalent content block becomes a signature, and so on, until wN flows in the original F have been covered. This greedy algorithm attempts to minimize the size of the set of signatures by choosing the most prevalent content block at each step.

We incorporate a *blacklisting* technique into signature generation. An administrator may configure Autograph with a blacklist of disallowed signatures, in an effort to prevent the system from generating signatures that will cause false positives. The blacklist is simply a set of strings. Any signature Autograph selects that is a substring of an entry in the blacklist is discarded; Autograph eliminates that content block from C without selecting it as a signature, and continues as usual. We envision that an administrator may run Autograph for an initial *training period*, and vet signatures with human eyes during that period. Signatures generated during this period that match common patterns in innocuous flows (*e.g.*, `GET /index.html HTTP/1.0`) can be added to the blacklist.

At the end of this process, Autograph reports the selected set of signatures. The current version of the system publishes signature byte patterns in Bro's signature format, for direct use in Bro. Table 3.1 summarizes the parameters that control Autograph's behavior.

3.3.3 Innocuous Traffic Included in Suspicious Flow Pool

Because the flow classifier heuristic is imperfect, innocuous flows will unavoidably be included in the signature generation process. We expect two chief consequences of their inclusion:

Prevalent signatures matching innocuous and malicious flows. One possible result is that the probabilistic COPP process will produce content blocks that contain only protocol header

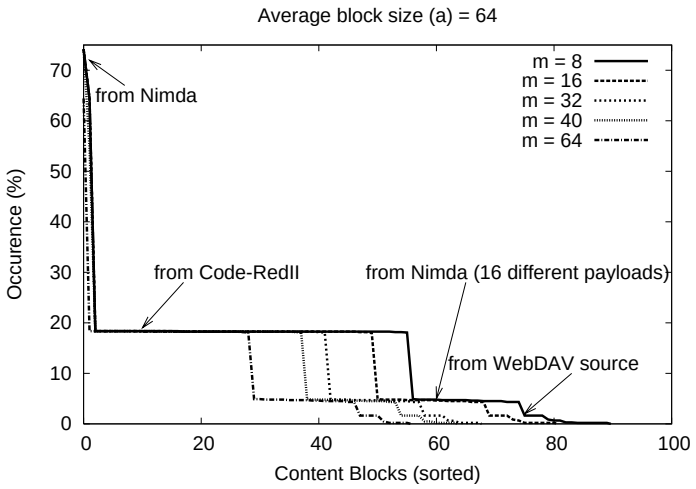


Figure 3.3: **Prevalence histogram of content blocks, a=64 bytes, ICSI2 DMZ trace, day 3 (24 hrs):** Content blocks from misclassified innocuous flows in the suspicious flow pool form the tail of the prevalence distribution, while content blocks from worm flows top the prevalence histogram.

or trailer data common to nearly *all* flows carrying that protocol, whether innocuous or malicious. Such blocks will top the prevalence histogram, but would clearly be abysmally unspecific if adopted for traffic filtering. To avoid choosing such unspecific content blocks, we can vary a and m toward longer block sizes.

Non-prevalent signatures for innocuous flows. Another possibility is that Autograph chooses a content block common to only a *few* innocuous flows. Such content blocks will not be prevalent, and will be at the tail of the prevalence histogram. Two heuristics can exclude these signatures from publication. First, by using a smaller w value, Autograph can avoid generation of signatures for the bottom $(1 - w)\%$ of the prevalence distribution, though this choice may have the undesirable side effect of delaying detection of worms. The second useful heuristic comes from our experience with the initial COPP implementation. Figure 3.3 shows the prevalence histogram Autograph generates from a real DMZ trace. Among all content blocks, only a few are prevalent (those from Code-RedII, Nimda, and WebDAV) and the prevalence distribution has a noticeable tail. We can restrict Autograph to choose a content block as a signature only if more than p flows in the suspicious flow pool contain it, to avoid publishing signatures for non-prevalent content blocks.

Symbol	Description
s	Port scanner detection threshold
a	COPP parameter: average content block size
m	COPP parameter: minimum content block size
M	COPP parameter: maximum content block size
w	Target percentage of suspicious flows to be represented in generated signatures
p	Minimum content block prevalence for use as signature
t	Duration suspicious flows held in suspicious flow pool
θ	Minimum size of suspicious flow pool to triggers signature generation process

Table 3.1: Autograph's signature generation parameters.

3.4 Evaluation: Local Signature Generation

We now evaluate the quality of signatures Autograph generates. In this section, we answer the following two questions: First, how does content block size affect the the sensitivity and specificity of the signatures Autograph generates? And second, how robust is Autograph to worms that vary their payloads?

Our experiments demonstrate that as content block size decreases, the likelihood that Autograph detects commonality across suspicious flows increases. As a result, as content block size decreases, Autograph generates progressively more sensitive but less specific signatures. They also reveal that small block sizes are more resilient to worms that vary their content, in that they can detect smaller common parts among worm payloads. Finally, they reveal that when run on real web traffic from a DMZ, Autograph with blacklisting can achieve zero false positives; can detect signatures for all worms present; and can quickly generate signatures for "established" worms with pre-existing populations of active infected hosts.

3.4.1 Offline Signature Generation on DMZ Traces

We first investigate the effect of content block size on the quality of the signatures generated by Autograph. In this subsection, we use a suspicious flow pool accumulated during an interval t of 24 hours, and consider only a single invocation of signature generation on that flow pool. No blacklisting is used in the results in this subsection, and filtering of content blocks that appear only from one source address before signature generation is disabled. All results we present herein are for a COPP Rabin fingerprint window of width $k = 4$ bytes.[4]

[4]We have since adopted a 16-byte COPP window in our implementation, to make it harder for worm authors to construct payloads so as to force particular content block boundaries; results are quite similar for

	IRP	ICSI	ICSI2
Measurement Period	Aug 1-7 2003	Jan 26 2004	Mar 22-29 2004
	1 week	24 hours	1 week
Inbound HTTP packets	70K	793K	6353K
Inbound HTTP flows	26K	102K	825K
HTTP worm sources	72	351	1582
scanned	56	303	1344
not scanned	16	48	238
Nimda sources	18	57	254
CodeRed II sources	54	294	997
WebDav exploit sources	-	-	336
HTTP worm flows	375	1396	7127
Nimda flows	303	1022	5392
CodeRed flows	72	374	1365
WebDav exploit flows	-	-	370

Table 3.2: Summary of traces.

In our experiments, we feed Autograph one of three packet traces from the DMZs of two research labs; one from Intel Research Pittsburgh (Pittsburgh, USA) and two from ICSI (Berkeley, USA). IRP's Internet link was a T1 at the time our trace was taken, whereas ICSI's is over a 100 Mbps fiber to UC Berkeley. All three traces contain the full payloads of all packets. The ICSI and ICSI2 traces only contain inbound traffic to TCP port 80, and are IP-source-anonymized. Both sites have address spaces of 2^9 IP addresses, but the ICSI traces contain more port 80 traffic, as ICSI's web servers are more frequently visited than IRP's.

For comparison, we obtain the full list of HTTP worms in the traces using Bro with well-known signatures for the Code-Red, Code-RedII, and Nimda HTTP worms, and for an Agobot worm variant that exploits the WebDAV buffer overflow vulnerability (present only in the ICSI2 trace). Table 3.2 summarizes the characteristics of all three traces.

Autograph's suspicious flow classifier identifies unsuccessful connection attempts in each trace. For the IRP trace, Autograph uses ICMP host/port unreachable messages to compile the list of suspicious remote IP addresses. As neither ICSI trace includes outbound ICMP packets, Autograph infers failed connection attempts in those traces by looking at incoming TCP SYN and ACK pairs.

We run Autograph with varied scanner detection thresholds, $s \in \{1, 2, 4\}$. These thresholds are lower than those used by Bro and Snort, in the interest of catching as many worm payloads as possible (crucial early in an epidemic). As a result, our flow classifier misclassi-

$k = 16$.

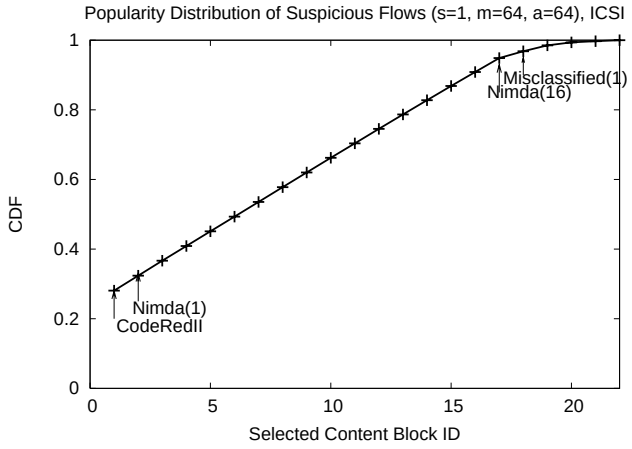


Figure 3.4: **Prevalence of Selected Content Blocks in Suspicious Flow Pool, ICSI DMZ trace (24 hrs):** Most flows are worms except for a few misclassified flows. Autograph generates signatures for every worm in the pool.

fies flows as suspicious more often, and more innocuous flows are submitted for signature generation.

We also vary the minimum content block size (m) and average content block size (a) parameters that govern COPP, but fix the maximum content block size (M) at 1024 bytes. We vary $w \in [10\%, 100\%]$ in our experiments. Recall that w limits the fraction of suspicious flows that may contribute content to the signature set. COPP adds content blocks to the signature set (most prevalent content block first, and then in order of decreasing prevalence) until one or more content blocks in the set match w percent of flows in the suspicious flow pool.

We first characterize the content block prevalence distribution found by Autograph with a simple example. Figure 3.4 shows the prevalence of content blocks found by COPP when we run COPP with $m = 64$, $a = 64$, and $w = 100\%$ over a suspicious flow pool captured from the full 24-hour ICSI trace with $s = 1$. At $w = 100\%$, COPP adds content blocks to the signature set until *all* suspicious flows are matched by one or more content blocks in the set. Here, the x axis represents the order in which COPP adds content blocks to the signature set (most prevalent first). The y axis represents the cumulative fraction of the population of suspicious flows containing any of the set of signatures, as the set of signatures grows. The trace contains Code-RedII, Nimda, and WebDAV worm flows. Nimda sources send 16 different flows with every infection attempt, to search for vulnerabilities under 16 different URLs. The first signature COPP generates matches Code-RedII; 28% of the suspicious flows are Code-RedII instances. Next, COPP selects 16 content blocks as signatures, one for each

of the different payloads Nimda-infected machines transmit. About 5% of the suspicious flows are misclassified flows. We observe that commonality across those misclassified flows is insignificant. Thus, the content blocks from those misclassified flows tend to be lowly ranked.

To measure true positives (fraction of worm flows found), we run Bro with the standard set of policies to detect worms (distributed with the Bro software) on a trace, and then run Bro using the set of signatures generated by Autograph on that same trace. The true positive rate is the fraction of the total number of worms found by Bro's signatures (presumed to find all worms) also found by Autograph's signatures.

To measure false positives (fraction of non-worm flows matched by Autograph's signatures), we create a *sanitized* trace consisting of all non-worm traffic. To do so, we eliminate all flows from a trace that are identified by Bro as worms. We then run Bro using Autograph's signatures on the sanitized trace. The false positive rate is the fraction of all flows in the sanitized trace identified by Autograph's signatures as worms.

Because the number of false positives is very low compared to the total number of HTTP flows in the trace, we report our false positive results using the *efficiency* metric proposed by Staniford *et al.* [126]. Efficiency is the ratio of the number of true positives to the total number of positives, both false and true. Efficiency is proportional to the number of false positives.

The graphs in Figure 3.5 show the sensitivity and the efficiency of the signatures generated by Autograph running on the full 24-hour ICSI trace for varied m. Here, we present experimental results for $s = 2$, but the results for other s are similar. Note that in these experiments, we apply the signatures Autograph generates from the 24-hour trace to the *same* 24-hour trace used to generate them.

The x axis varies w. As w increases, the set of signatures Autograph generates leads to greater sensitivity (fewer false negatives). This result is expected; greater w values cause Autograph to add content blocks to the signature set for an ever-greater fraction of the suspicious flow pool. Thus, if a worm appears rarely in the suspicious flow pool, and thus generates non-prevalent content blocks, those blocks will eventually be included in the signature set, for sufficiently large w.

However, recall from Figure 3.4 that about 5% of the suspicious flows are innocuous flows that are misclassified by the port-scanner heuristic as suspicious. As a result, for $w > 95\%$, COPP risks generating a less specific signature set, as COPP begins to select content blocks from the innocuous flows. Those content blocks are most often HTTP trailers, found in common across misclassified innocuous flows.

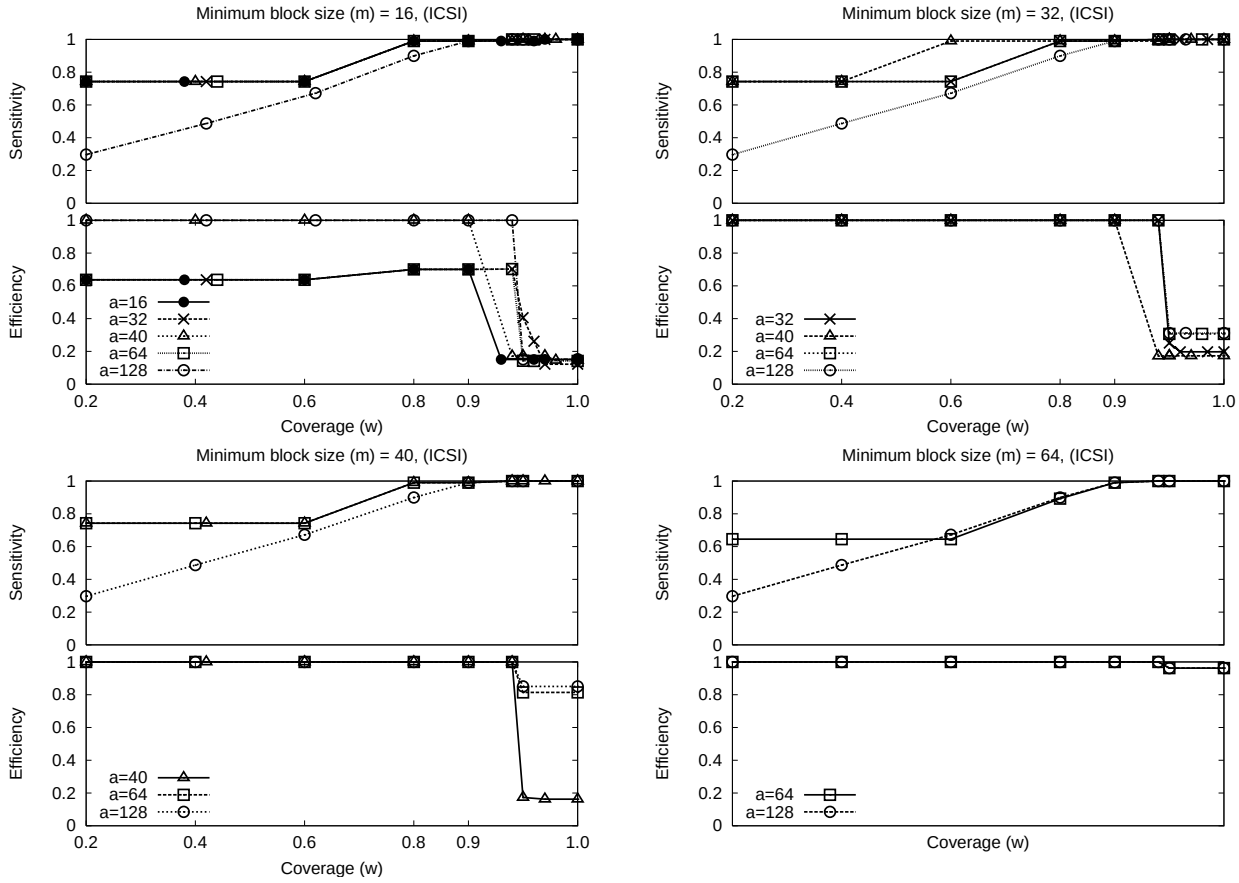


Figure 3.5: **Sensitivity and Efficiency of Selected Signatures, ICSI DMZ trace (24 hrs):** Sensitivity represents the fraction of true positives. Efficiency represents the fraction of true positives to all flagged flows.

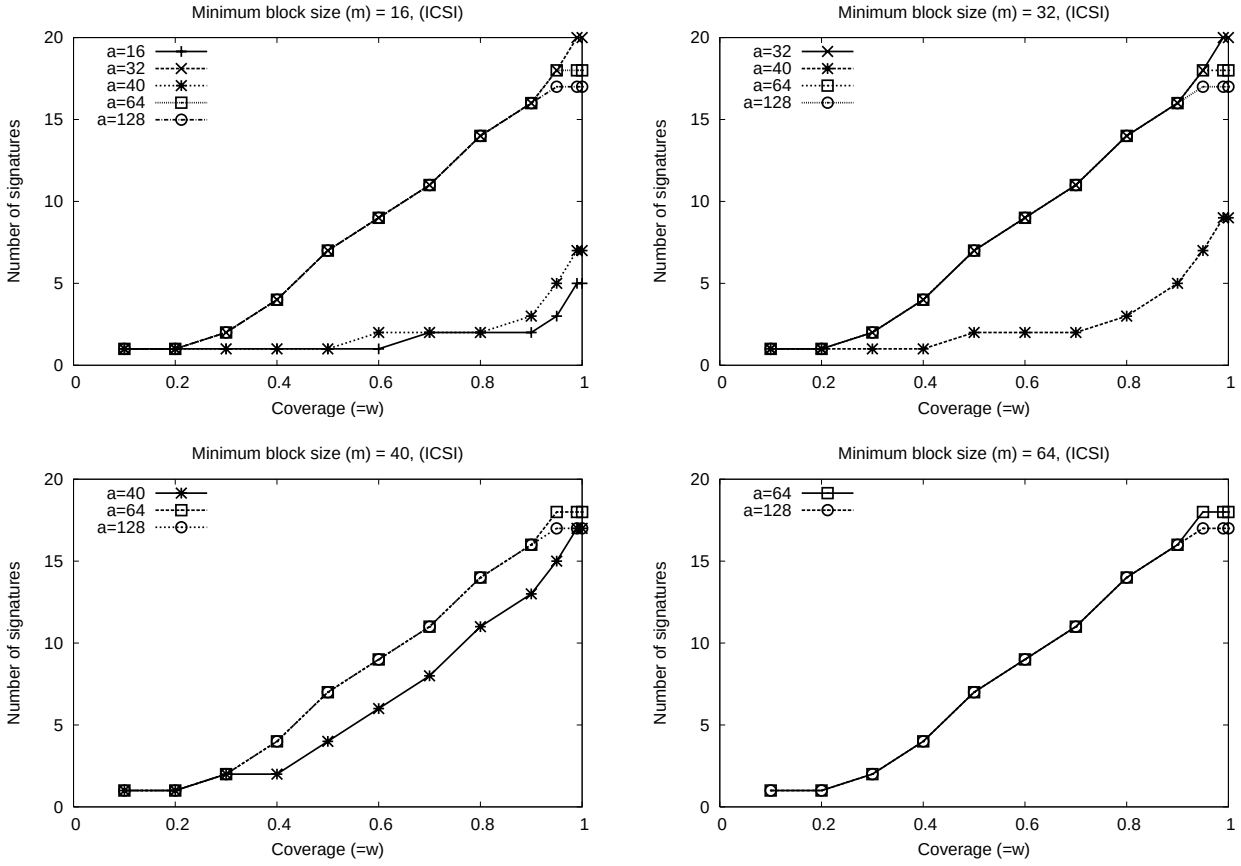


Figure 3.6: **Number of Signatures, ICSI DMZ trace (24 hrs):** For IDS efficiency, a smaller number of signatures is preferred.

For this trace, COPP with $w \in [90\%, 94.8\%]$ produces a set of signatures that is *perfect:* it causes 0 false negatives and 0 false positives. Our claim is *not* that this w parameter value is valid for traces at different sites, or even at different times; on the contrary, we expect that the range in which no false positives and no false negatives occurs is sensitive to the details of the suspicious flow population. Note, however, that the existence of a range of w values for which perfect sensitivity and specificity are possible serves as a very preliminary validation of the COPP approach—if no such range existed for this trace, COPP would always be forced to trade false negatives for false positives, or vice-versa, for *any* w parameter setting. Further evaluation of COPP on a more diverse and numerous set of traffic traces is clearly required to determine whether such a range exists for a wider range of workloads.

During examination of the false positive cases found by Autograph-generated signatures when $w > 94.8\%$, we noted with interest that Autograph's signatures detected Nimda sources *not* detected by Bro's stock signatures. There are only three stock signatures used by Bro to spot a Nimda source, and the Nimda sources in the ICSI trace did not transmit those particular payloads. We removed these few cases from the count of false positives, as Autograph's signatures *correctly* identified them as worm flows, and thus we had *erroneously* flagged them as false positives by assuming that any flow not caught by Bro's stock signatures is not a worm.

We now turn to the effect of content block size on the specificity and the number of signatures Autograph generates. Even in the presence of innocuous flows misclassified as suspicious, the largest average and minimum content block sizes (such as 64 and 128 bytes) avoid most false positives; efficiency remains close to 1. We expect this result because increased block size lowers the probability of finding common content across misclassified flows during the signature generation process. Moreover, as signature length increases, the number of innocuous flows that match a signature decreases. Thus, choosing larger a and m values will help Autograph avoid generating signatures that cause false positives.

Note, however, there is a trade-off between content block length and the number of signatures Autograph generates, too. For large a and m, it is more difficult for COPP to detect commonality across worm flows unless the flows are identical. So as a and m increase, COPP must select more signatures to match any group of variants of a worm that contain some common content. The graphs in Figure 3.6 present the size of the signature set Autograph generates as a function of w. For smaller a and m, Autograph needs fewer content blocks to cover w percent of the suspicious flows. In this trace, for example, COPP can select a short byte sequence in common across different Nimda payload variants (*e.g.*, `cmd.exe?c+dir HTTP/1.0..Host:www..Connection: close....`) when we use small a and m, such

as 16. The size of the signature set becomes a particular concern when worms aggressively vary their content across infection attempts, as we discuss in the next section. Before continuing on, we note that results obtained running Autograph on the IRP and ICSI2 traces are quite similar to those reported above.

3.4.2 Polymorphic and Metamorphic Worms

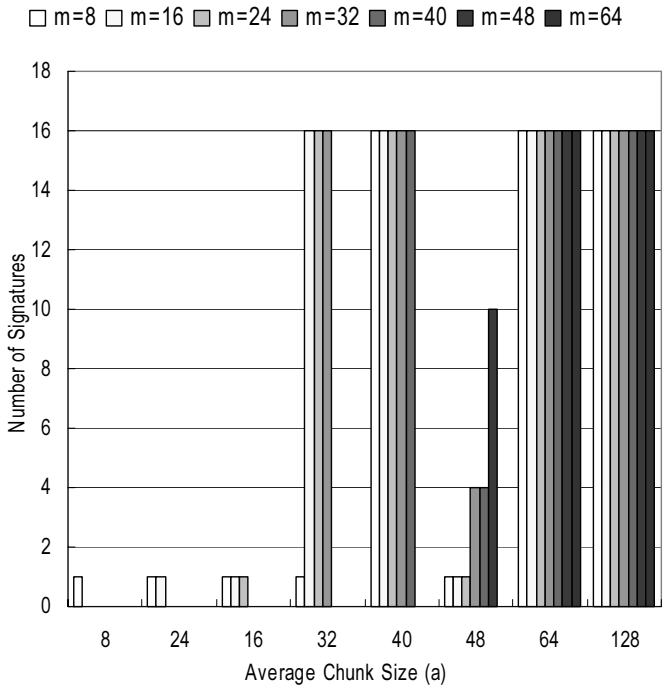


Figure 3.7: **Content block size *vs.* number of signatures:** Autograph finds a short signature that match all instances of a polymorphic worm if content block size parameters (*m*: minimum content block size) are set to small values.

We expect short content blocks to be most robust against worms that vary their content, such as polymorphic worms, which encrypt their content differently on each connection, and metamorphic worms, which obfuscate their instruction sequences on each connection. Unfortunately (fortunately?) no such Internet worm has yet been reported in the wild. To test Autograph's robustness against these varying worms, we generate a synthetic polymorphic worm based on the Code-RedII payload. A Code-RedII worm payload consists of a regular HTTP GET header, more than 220 filler characters, a sequence of Unicode, and the main worm executable code. The Unicode sequence causes a buffer overflow and transfers execution flow to the subsequent worm binary. We use *random values* for all filler bytes, and even for the worm code, but leave the HTTP GET command and 56-byte Unicode sequence

fixed. This degree of variation in content is more severe than that introduced by the various obfuscation techniques discussed by Christodorescu *et al.* [25]. As shown in Figure 3.7, when a relatively short, invariant string is present in a polymorphic or metamorphic worm, Autograph can find a short signature that matches it, when run with small average and minimum content block sizes. However, such short content block sizes may be unspecific, and thus yield signatures that cause false positives.

3.5 P2P Traffic and Signature Generation

Since the early 2000s, peer-to-peer (P2P) networks have been widely used in large file sharing application development, and the volume of P2P application traffic continues to grow [66]. P2P applications use many-to-many connections to distribute data efficiently, which leads to similarity between P2P application traffic and worm traffic in terms of content prevalence and wide-spreading behavior. This similarity could introduce significant amount of noise into Autograph.

In this section, we analyze the effect of P2P application traffic on Autograph. In particular, we are interested in answering the following three questions.

- How frequently does Autograph generate false signatures due to P2P application traffic?
- How similar is P2P traffic with worm traffic? And,
- how do we reduce the false positives caused by P2P application traffic?

To answer the questions, we test Autograph with a trace collected at a gateway of a university campus network. We will describe the details of the trace in Section 3.5.1 as we report our observation.

Our experiments show that Autograph does generate a few unspecific signatures from P2P file sharing application traffic. Even though the number of those unspecific signatures is small, the use of those unspecific signatures may interrupt one of the popular applications. Our investigation of P2P application traffic shows that traffic from many popular P2P file sharing applications mimics the traffic characteristics of worm traffic. Moreover, it may be difficult to avoid generating unspecific signatures from P2P traffic if automated signature generation systems rely only on content invariance and wide-spreading behavior analysis. Combining other worm detection heuristics such as port scanner detection helps avoid generating unspecific signatures.

3.5.1 Empirical Results: False Positives

We first examine the quality of the signatures Autograph generates in an *online* setting and the number of signatures generated from P2P application traffic. In an online deployment, Autograph holds suspicious flows observed during the last t minutes, provided that the number of suspicious flows accumulated exceeds θ.

The suspicious flow pool stores all suspicious flows observed during the last t minutes. If t is too short, it is unlikely that the pool will contain enough flows to generate a signature for a slowly propagating worm early in its propagation. Too long a t is also undesirable, as the number of flows Autograph has to store and process will be large. Moreover, the longer t is, the more likely the pool will include content blocks from innocuous flows from multiple sources. We enable filtering of content blocks that originate at only one source IP address before signature generation, as described in Section 3.3. In this experiment, we consider t = 10 minutes, which is long enough for this network to generate signatures for the CodeRedI-v2 worm before the worm infects 5% of vulnerable hosts in the Internet.[5] We set w and s to be $w = 94\%$ and $s = 2$ respectively. We set the average content block size $a = 64Bytes$.

In our experiments, we feed Autograph an 1-hour traffic payload trace collected from a campus network gateway in August 2005. The network has one class-B address block and one class-C address block. The trace was captured with a DAG-4 monitor and packet capture software [44] installed on an 1 Gbps interface at the border gateway. The gateway has three 1 Gbps interfaces and performs load balancing. Thus, the trace capture one third of incoming traffic toward the campus. The trace includes 376 millions of IP packets (i.e., 185 GB in byte volume). We check the trace with `bro` and its worm signature definitions [102], and find none of well-known recent worms are present in the trace.

With the above parameter setting, Autograph generates four 'unspecific' signatures: two signatures from port 80, and two other signatures from port 9493. Figure 3.8 shows those four payload pattern strings with the port numbers. Port 9493 is used by a P2P-based file sharing system (FileGuri [6]) popular in Asia. Port 80 is a well-known web port. Both ports use the HTTP-based protocol, so we could easily find where this payload pattern belongs in each flow. `SIG1` specifies the application name and its version number, and they appear in the HTTP header portion of every control message of the P2P system. `SIG3` and `SIG4` are from the HTTP headers of messages. Understanding the structure of the HTTP protocol,

[5]The measurement window length t is computed based on simulation, where every infected host scans at the rate of 358 scans/minute and 360,000 vulnerable hosts exist. The campus network has more than 3300 hosts with port 80 open.

[6]FileGuri (Korean P2P file sharing application) `http://www.fileguri.com`

we can use signature blacklist heuristic for those three signatures. SIG2 is a data query message. The values for each argument in the query change over time, so it is difficult to exclude this type of signatures from Autograph with the static signature blacklist heuristic.

```
SIG1 (port 9493) *\/*\x0d\x0aUser-Agent: Freechal P2P\/1.0\x0d\x0aP2P-Authen

SIG2 (port 9493) GET \/?p2pmethod=search&keyword=%XX%XX%XX%XX&extension=&time
                 out=2000&searchdownload=1&nowaiting=0 HTTP\/1.1\x0d\x0aAccep
                 t: *\/*\x0d\x0aUser-Agent: Freechal P2P\/1.0\x0d\x0aP2P-Authen

SIG3 (port 80)   ozilla\/5.0(compatible; Yahoo! Slurp; http:\/\/help.yahoo.com
                 \/help\/us\/ysearch\/s
SIG4 (port 80)   lurp)\x0d\x0aAccept-Encoding: gzip, x-gzip\x0d\x0a\x0d\x0a
```

Figure 3.8: **Unspecific Signatures Generated by Autograph:** port 9493 is used by a P2P-based file sharing system. The system uses HTTP. SIG1 belongs to the header of messages that indicates the version of application software. SIG2 is from a query message. Private parts are sanitized with XX. SIG3 is from a web search crawler, and SIG4 is from HTTP ending. Except the SIG2, generating signature blacklist for them is not difficult.

Autograph generated only a few signatures in our experiment and most of them can be excluded by signature blacklisting. However, Autograph generates signatures from P2P traffic that are not easily suppressed with static signature blacklist entries.

3.5.2 P2P Application Traffic Properties

We examine P2P traffic in terms of content prevalence, address dispersion, and burstiness. If those properties are strongly similar to that of a worm, it is not impossible for Autograph to generate many signatures from P2P traffic. We extracted P2P application traffic using the technique proposed by Karagiannis et al. [66] along with the list of well-known P2P applications ports shown in Figure 3.9.

Content Prevalence.

Most popular P2P file sharing applications (i.e, eDonkey and FileGuri) in our traces tend to send similar messages to peers because of their protocol design, and thus, interfere with content prevalence measurement. To find how frequently P2P applications exchange similar payloads, we extracted all 48-byte overlapping substrings from each packet and counted the occurrence in 10 minute measurement interval. Because the number of substrings is huge, we randomly sampled them with a probability 1/64. Figure 3.10 shows the cumulative

Name	Protocol	Ports	Name	Protocol	Ports
eDonkey [42]	tcp udp	4662-4667, 4242 4662-4665, 4672	P2P-Radio [99]	tcp udp	2000, 2222 2010
iMesh [62]	tcp	5000	vShare [135]	tcp	8404
Soribada [124]	udp	7674-7675, 22321	Blubster [9]	udp	41170
WinMX [144]	tcp udp	6699 6257	Shareshare [117]	tcp udp	6599 6777
BitTorrent [7]	tcp	6881-6889	DirectConnect [39]	tcp,udp	411-412
Kazaa [68]	tcp	1214	Napster [92]	tcp	6600-6699
Gnutella [29]	tcp,udp	6346-6347	Clubbox [97]	tcp	19101
Fileguri [48]	tcp	9493	Madster(AIMster) [80]	tcp	23172, 9922
HotLine [59]	tcp,udp	5500–5503	GoBoogy [54]	tcp,udp	5335
MP2P [90]	tcp,udp	41170			

Figure 3.9: **Ports used by P2P applications**

distribution function of payload patterns (substrings) for each P2P application traffic. There are frequently occurring payload patterns. Because 'query' messages are multicast to all nodes, 7.5% of FileGuri and 20% of eDonkey payload patterns appeared more than once during the measurement period. The values in the legend show the maximum number of occurrences observed. Even when a P2P application is designed to avoid such common message exchanges, popular files distributed over the P2P network can potentially result in the file content downloaded by many hosts during their peak periods of popularity. In particular, the file striping techniques that BitTorrent and other P2P file sharing systems employ, worsen the situation.

Address Dispersion

P2P file sharing applications take advantage of many-to-many communication to speed file downloading and improve availability of data. We now examine the fan-out of P2P hosts and non-P2P hosts based on the identified connection type. Figure 3.11 compares the number of external peers of internal P2P hosts and non-P2P hosts for a 5 minute time period in the trace.

Note that the same host can be both a P2P and non-P2P host. Not surprisingly, about 50% of P2P hosts have more than 30 peers, while most non-P2P hosts tend to have fewer than 10 peers. A significant fraction ($>$ 10%) of P2P hosts are contacted by thousands of external hosts for 10 minutes. Less than 1.1% of non-P2P hosts communicate with more than 10 peers and less than 0.2% of non-P2P hosts have more than a hundred peers. Except for a few popular web servers, game hosts, and DNS servers, those non-P2P hosts with hundreds

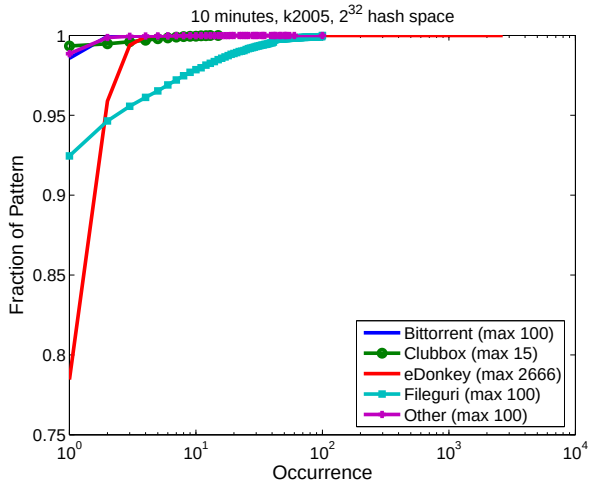


Figure 3.10: **CDF of Payload Pattern Strings for Each P2P Application:** there are non-negligible fraction of frequently appearing P2P payload pattern strings. Because of multicast search requests, more than 5% of patterns from FileGuri traffic appear more than once. eDonkey patterns have less similarity across requests, but one of the payload patterns in eDonkey traffic appeared 2666 times.

of peers also have well-known P2P ports open, so we assume that those connections also belong to P2P communications.

Figure 3.12 shows the address dispersion of payload patterns from popular P2P systems. Each dot represents a payload pattern string that appeared more than 30 times in a 10 minute interval. The position of a dot represents the number of distinct source IPs and destination IPs that send the corresponding payload pattern string. Based on the popularity and the content distribution mechanism, P2P applications show different patterns in the number of source and destination IPs. However, all four graphs show that there are payload patterns that satisfy both content prevalence and address dispersion. The payload patterns that are sent to hundreds of destinations by tens of sources result in false signature generation because they satisfy the address dispersion criteria even when a content-based signature generation system selects payload pattern strings exchanged between hundreds sources and destinations.

Burstiness

When a P2P client requests a file search or a download of file segments, the client sends out the request messages to multiple peers. During the peak periods of popular files, the same payload patterns are sent by multiple clients within a short period of time, and those patterns will satisfy the content prevalence and address dispersion criteria.

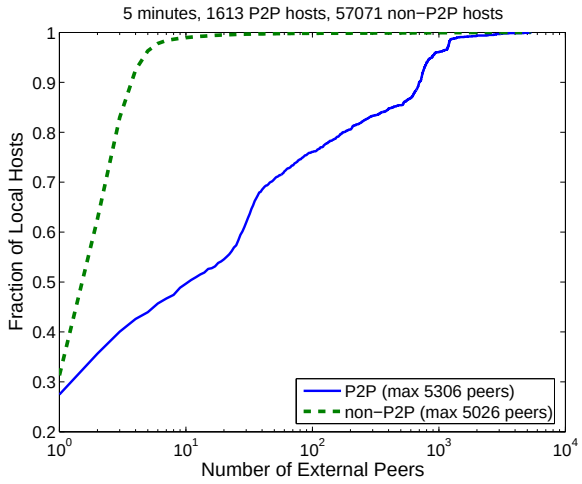


Figure 3.11: **Number of external peers of P2P hosts and non-P2P hosts:** CDF of internal hosts based on the number of external peers. P2P hosts tend to communicate with many hosts. More than 50% of P2P hosts were contacted by more than 10 external hosts for 5 minutes, while most non-P2P hosts were contacted by less than ten external hosts. This trend could look like the address dispersion property of worm traffic.

We examine the fraction of P2P traffic in our data sets. P2P applications contribute to more than 40% of the traffic, and this traffic is increasing over time. We compared the fraction of P2P traffic measured at the same gateway. In 2004, known P2P traffic accounts for 39% of total byte volume and 31% of total packet counts. In 2006, the fraction of P2P traffic reached 42% of byte volume and 50% of packet counts. Note that our P2P traffic identification in this measurement may miss some P2P traffic and our result underestimates the fraction of actual P2P applications. We observe that more applications are adopting P2P techniques, and provide options for users to choose the communication port. As P2P communication becomes more popular and more proprietary P2P protocols penetrate, pattern-extraction-based signature generation will become more difficult because P2P traffic may mimic worm behavior.

3.5.3 Preventing Unspecific Signature Generation from P2P Traffic

Despite many similarities between P2P and worm traffic, Autograph generates few signatures. This is because Autograph combines suspicious flow classification heuristics, such as port-scanner's flow detection, with content prevalence and wide-spreading behavior measurement. However, it is not impossible for a P2P host to behave like a port-scanner. For example, in a BitTorrent network, a central server (called a tracker) maintains a list of clients

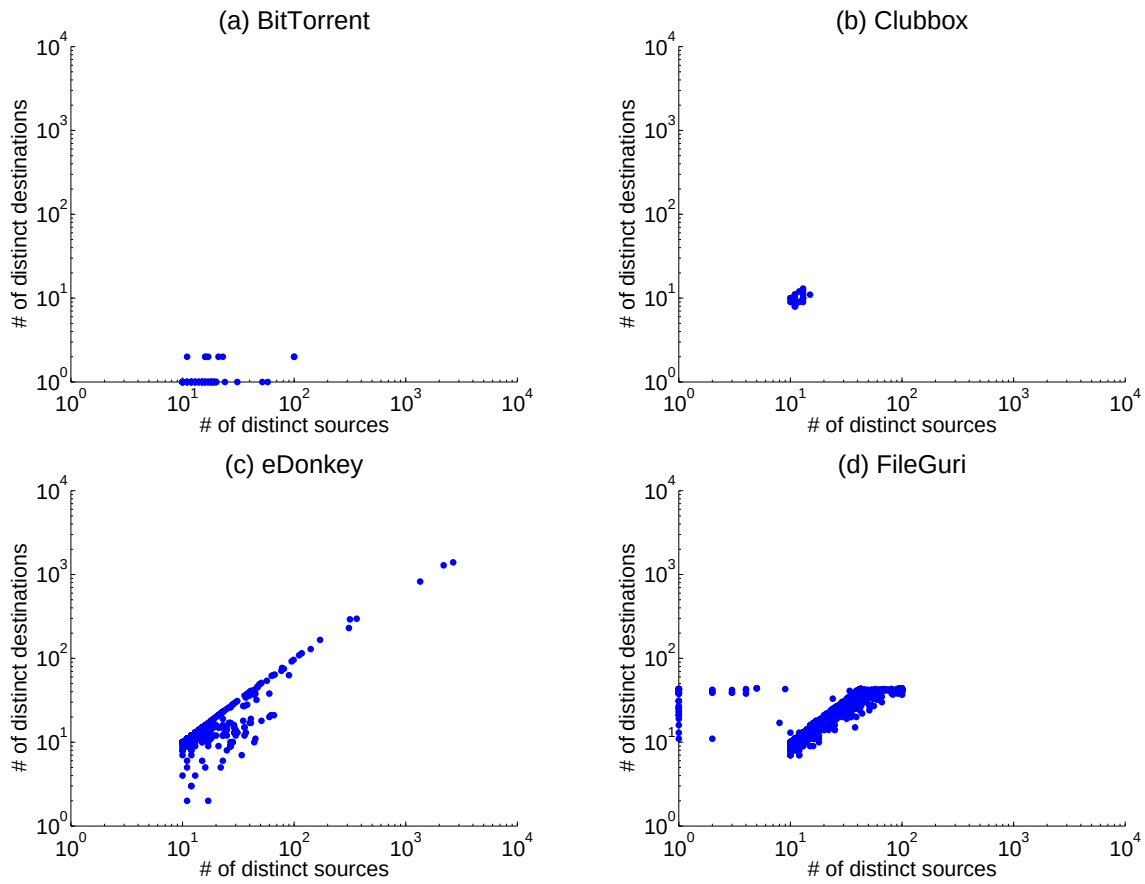


Figure 3.12: **Distinct source and destination counts per applications:** Each dot represents the number of distinct sources (x-axis) and destinations (y-axis) of a payload pattern that appeared more than 10 times during a 10 minute time interval. Payload patterns exchanged between many hosts satisfy address dispersion and content prevalence criteria. Thus, signature generation systems that rely only on the two properties will generate 'unspecific' signatures based on the payload patterns.

currently downloading a file, and clients who want the same file download segments of the file from the listed clients. The list may be stale, causing some clients to contact inactive clients. In this case, the requesting client IP addresses are classified as port scanners and other requests from the IPs are considered to be suspicious. The common payload patterns from BitTorrent message headers and popular file hashes meet the content prevalence and address dispersion criteria, resulting in generated signatures. There are two heuristics Autograph uses to avoid such misclassification.

Bookkeeping Active Servers: Autograph can keep track of server hosts. When a server becomes unavailable, Autograph does not count port-scans toward the server's IP address for a while. However, this may result in delay in the real worm source IP address detection, and thus cause delay in worm signature detection.

Higher Port Scanner Detection Threshold for P2P Ports: Autograph can choose a higher threshold s for P2P traffic monitoring. For example, when we use the scanner detection threshold $s \in [3, 4]$ on the same trace, Autograph does not generate any signature from P2P traffic. Similarly, combining better worm flow selection heuristics with content prevalence analysis reduces the false positive rate. However, no heuristic is perfect; Heuristics can introduce more false negatives. Thus, choosing heuristics must be done carefully.

3.6 Attacks and Limitations

We catalog a few attacks that one might mount against Autograph, and limitations of the current system.

Overload. Autograph reassembles suspicious TCP flows. Flow reassembly is costly in state in comparison with processing packets individually, but defeats the subterfuge of fragmenting a worm's payload across many small packets [102]. We note that the number of inbound flows a monitor observes may be large, in particular after a worm spreads successfully. If Autograph tries to reassemble every incoming suspicious flow, it may be susceptible to DoS attack. We note that Autograph treats all destination ports separately, and thus parallelizes well across ports; a site could run multiple instances of Autograph on separate hardware, and thus increase its aggregate processing power, for flow reassembly and all other processing. Autograph may also sample suspicious flows when the number of suspicious flows to process exceeds some threshold; we intend to investigate this heuristic in future.

Source-address-spoofed port scans. Port scans from spoofed IP source addresses are a peril for most IDSes. The chief reason for monitoring port scans is to limit the damage their originators can inflict, most often by filtering packets that originate from known port scanners. Such filtering invites attackers to spoof port scans from the IP addresses of those whose traffic they would like to block [102, 64]. Source-spoofed port scans can be used to mount a new type of Denial-of-Service (DoS) attacks. A source-spoofing attacker could cause a remote source's traffic to be included by Autograph in signature generation.

Fortunately, a simple mechanism holds promise for rendering both these attacks ineffective. Autograph classifies an inbound SYN destined for an unpopulated IP address or port with no listening process as a port scan. To identify TCP port scans from spoofed IP source addresses, an Autograph monitor could respond to such inbound SYNs with a SYN/ACK, provided the router and/or firewall on the monitored network can be configured not to

respond with an ICMP host or port unreachable. If the originator of the connection responds with an ACK with the appropriate sequence number, the source address on the SYN could not have been spoofed. The monitor may thus safely view all source addresses that send proper ACK responses to SYN/ACKs as port scanners. Non-ACK responses to these SYN/ACKs (RSTs or silence) can then be ignored; *i.e.,* the source address of the SYN is not recorded as a port scanner. Note that while a non-source-spoofing port scanner may *choose* not to respond with an ACK, any source that hopes to complete a connection and successfully transfer an infecting payload *must* respond with an ACK, and thus identify itself as a port scanner. Jung *et al.* independently propose this same technique in [64].

Worms with scanning suppression techniques. If a worm employes scanning suppression techniques, such as hit-list scanning (i.e a *Warhol* worm), permutation scanning (i.e. worms that use self-coordinating scanning), topological scanning (i.e. worms that harvest the next victims' IP addresses from already infected victim), internal-sized hit list scanning (i.e. a *flash* worm) [127], rather than randomly scans IP addresses that may or may not correspond to listening servers, Autograph's port-scan-based suspicious flow classifier will fail utterly to include that worm's payloads in signature generation. Malicious payload gathering methods such as honeypots are similarly stymied by hit-list worm propagation. Nevertheless, any innovation in the detection of flows generated by hit-list based worms may be incorporated into Autograph, to augment or replace the naive port-scan-based heuristic used in our prototype. For example, we envision that Autograph captures all the flows in and out of the potential bot hosts detected by the hit-list worm detection algorithm described in Collins and Reiter's work [31], and applies payload analysis against all the flows to find payload-based signatures of the worm.

3.7 Summary

We have presented Autograph, a system that automatically generates signatures for novel Internet worms that propagate using TCP transport and perform random scanning. Autograph generates signatures by analyzing the popularity of portions of flow payloads, without requiring knowledge of application protocols above the TCP layer. We evaluated the prototype of Autograph with a real DMZ trace and showed that Autograph can produce signatures that exhibit high sensitivity and high selectivity. P2P application traffic shares many similarities with worm traffic, which makes purely content-based signature generation difficult to produce signatures specific to worms. Combination with suspicious flow

selection heuristics helps reduce false positives caused by P2P application, but the choice of such heuristics may narrow the range of detectable worms, and must be made carefully.

Chapter 4

Distributed Signature Generation with Port-Scanner List Sharing

At the start of a worm's propagation, the aggregate rate at which all infected hosts scan the monitored IP address space is quite low. Because Autograph relies on overhearing unsuccessful scans to identify suspicious source IP addresses, early in an epidemic an Autograph monitor will be slow to accumulate suspicious addresses, and in turn slow to accumulate worm payloads. In this chapter, we examine Autograph's speed in detecting a signature for a *new* worm after the worm's release and utilize distributed port-scanner detection to expedite Autograph's signature detection. The main contributions of our work are summarized as follows:

- We demonstrate that operating multiple, distributed instances of Autograph significantly speeds up this process, compared to running a single instance of Autograph on a single edge network. We use a combination of simulation of a worm's propagation and DMZ-trace-driven simulation to evaluate the system in the online setting; our sense of ethics restrains us from experimentally measuring Autograph's speed at detecting a novel worm *in vivo.*

- We propose the *tattler* mechanism that allows distributed Autograph monitors to tattle to one another about port scanners without overloading the network even when the number of port scanners increases due to the worm propagation.

Measuring how quickly Autograph detects and generates a signature for a newly released worm is important because it has been shown in the literature that successfully containing a worm requires early intervention. Recall that Provos' results [105] show that reversing an epidemic such that fewer than 50% of vulnerable hosts ever become infected can

require intervening in the worm's propagation before 5% of vulnerable hosts are infected. Two delays contribute to the total delay of signature generation:

- How long must an Autograph monitor wait until it accumulates enough worm payloads to generate a signature for that worm?
- Once an Autograph monitor receives sufficient worm payloads, how long will it take to generate a signature for the worm, given the background "noise" (innocuous flows misclassified as suspicious) in the trace?

We first compare the signature detection speed when we rely on a single Autograph monitor and utilize multiple, distributed instances of Autograph monitors in Section 4.1. Motivated by the analysis, we propose, in Section 4.2, an extension of Autograph that allows distributed monitors to collaboratively identify suspicious addresses and speeds up suspicious address identification. We measure the performance of the distributed Autograph in terms of its bandwidth consumption and worm payload accumulation speed in Section 4.3 and Section 4.4, respectively. In Section 4.5, we evaluate the signature detection time when a Code-RedI-v2-like worm propagates. Our results elucidate the fundamental trade-off between early generation of signatures for novel worms and the specificity of these generated signature. After discussing the value of distributed monitoring in a heterogeneous network environment and the possible attacks against distributed Autograph in Section 4.6, we summarize our findings in Section 4.7.

4.1 Single *vs.* Multiple Monitors

The probability of observing probes from a worm-infected host is proportional to the size of monitored network, D, and the fraction of unoccupied IP addresses in the network, r. The probability is $\frac{D \cdot r}{\Omega}$ when Ω is the size of the worm's scanning space. For a randomized scanning worm with the probe rate s, we can model the number of scans as a Poisson process. The probability that s_i scans hit the monitor during a time interval of length t is $P_{s_i}(t) = \frac{e^{-\alpha t} \cdot (\alpha t)^{s_i}}{s_i!}, \alpha = s \cdot \frac{Dr}{\Omega}$. Thus, the probability that the Autograph monitor receives more than or equal to s scans from a worm-infected source is $1 - \sum_{s_i=0}^{s-1} P_{s_i}(t)$.

For example, let us assume that our Autograph monitor monitors a /16 IP address space, 90% of the IP addresse space is not occupied, and a worm-infected host scans the 2^{32} IP address space with the scanning rate $s = 600$ probes/minute. Then the probability to catch at least 2 probes from a single host ($s = 2$) is only 0.44, even 3 hours after the host is infected.

To understand the signature detection speed during a worm propagation, we simulate a Code-RedI-v2-like worm, which is after that of Moore *et al.* [89]. We simulate a vulnerable population of 338,652 hosts, the number of infected source IPs observed in [85] that are uniquely assignable to a single Autonomous System (AS) in the BGP table data (obtained from University of Oregon Route Views Project [1]) of the 19th of July, 2001, the date of the Code-Red outbreak. There are 6378 ASes that contain at least one such vulnerable host in the simulation. Unlike Moore *et al.*, we do not simulate the reachability among ASes in that BGP table; we make the simplifying assumption that all ASes may reach all other ASes. This assumption may cause the worm to spread somewhat faster in our simulation than in Moore *et al.*'s.

We assign actual IP address ranges for real ASes from the BGP table snapshot to each AS in the simulation, according to a truncated distribution of the per-AS IP address space sizes from the entire BGP table snapshot. The distribution of address ranges we assign is truncated in that we avoid assigning any address blocks larger than /16s to any AS in the simulation. We avoid large address blocks for two reasons: first, few such monitoring points exist, so it may be unreasonable to assume that Autograph will be deployed at one, and second, a worm programmer may trivially code a worm to avoid scanning addresses within a /8 known to harbor an Autograph monitor. Our avoidance of large address blocks only lengthens the time it will take Autograph to generate a worm signature after a novel worm's release.

We assume 50% of the address space within the vulnerable ASes is populated with reachable hosts, that 25% of these reachable hosts run web servers, and we fix the 338,652 vulnerable web servers uniformly at random among the total population of web servers in the simulation.

Finally, the simulated worm propagates using random IP address scanning over the entire routable IPv4 address space, and a probe rate of 10 probes per second. We simulate network and processing delays, randomly chosen in $[0.5, 1.5]$ seconds, between a victim's receipt of an infecting connection and its initiation of outgoing infection attempts. We begin the epidemic by infecting 25 vulnerable hosts at time zero. Figure 4.1 shows the growth of the epidemic within the vulnerable host population over time. After 135 minutes, the worm infects 10% of vulnerable hosts, and after 15 minutes from then, it infects 25% of vulnerable hosts.

In these first simulations, we place Autograph monitors at a randomly selected 1% of the ASes that include vulnerable hosts (63 monitors). Figure 4.2 shows the maximum and

[1] `http://www.routeviews.org`

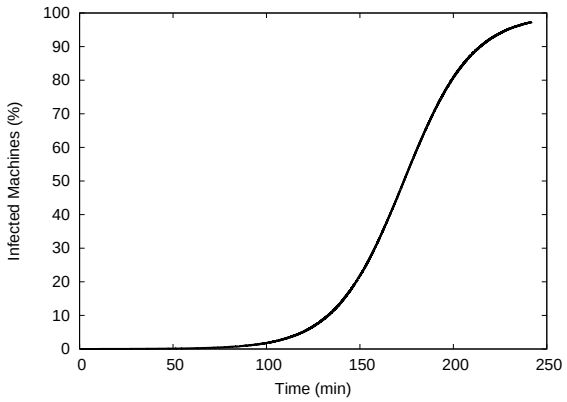


Figure 4.1: **Infection progress for a simulated Code-RedI-v2-like worm.**

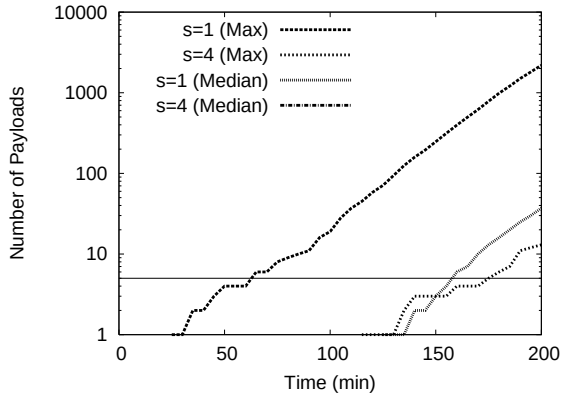


Figure 4.2: **Payloads observed over time:** single, isolated monitors.

median numbers of payloads detected over time across all monitors; note that the y axis is log-scaled. First, let us consider the case where only a single site on the Internet deploys Autograph on its network. In this case, it is the median time required by all 63 monitors to detect a given number of flows that approximates the expected time for a singleton monitor to do the same. When monitors identify port scanners aggressively, after a single failed connection from a source address ($s = 1$), the median monitor accumulates 5 worm payloads after over 150 minutes. Using the more conservative port-scan threshold $s = 4$, the median monitor accumulates *no* payloads within 167 minutes. These results are not encouraging—from Figure 4.1, we know that after 150 minutes, over 25% of vulnerable hosts have been infected.

Now let us consider the case where 63 monitors are used. If we presume that the first monitor to generate a signature for the worm may (nearly) instantly disseminate that signature to all who wish to filter worm traffic, by application-level multicast [18] or other means,

the earliest Autograph can possibly find the worm's signature is governed by the "luckiest" monitor among the 63 monitors—the first one to accumulate the required number θ of worm payloads. The "luckiest" monitor in this simulated distributed deployment detects 5 worm payloads shortly before 66 minutes have elapsed. This result is far more encouraging—after 66 minutes, fewer than 1% of vulnerable hosts have been infected. Thus, provided that all Autograph monitors disseminate the worm signatures they detect in a timely fashion, there is immense benefit in the speed of detection of a signature for a novel worm when Autograph is deployed distributedly, even at as few as 1% of ASes that contain vulnerable hosts.

Using the more conservative port-scan threshold $s = 4$, the monitor in the distributed system to have accumulated the most worm payloads after 10000 seconds has still only collected 4. Here, again, we observe that targeting increased specificity (by identifying suspicious flows more conservatively) comes at a cost of reduced sensitivity; in this case, sensitivity may be seen as the number of worm flows matched *over time.*

Running multiple independent Autograph monitors clearly pays a dividend in faster worm signature detection. A natural question that follows is whether detection speed might be improved further if the Autograph monitors shared information with one another in some way.

4.2 tattler: Distributed Gathering of Suspect IP Addresses

We now introduce an extension to Autograph named *tattler* that, as its name suggests, shares suspicious source addresses among all monitors, toward the goal of accelerating the accumulation of worm payloads.

We assume in the design of tattler that a multicast facility is available to all Autograph monitors, and that they all join a single multicast group. While IP multicast is not a broadly deployed service on today's Internet, there are many viable end-system-oriented multicast systems that could provide this functionality, such as Scribe [18]. In brief, Autograph monitor instances could form a Pastry overlay, and use Scribe to multicast to the set of all monitors. We further assume that users are willing to publish the IP addresses that have been port scanning them.[2]

The tattler protocol is essentially an application of the RTP Control Protocol (RTCP) [115], originally used to control multicast multimedia conferencing sessions, slightly extended for use in the Autograph context. The chief goal of RTCP is to allow a set of senders who all sub-

[2]In cases where a source address owner complains that his address is advertised, the administrator of an Autograph monitor could configure Autograph not to report addresses from the uncooperative address block.

scribe to the same multicast group to share a capped quantity of bandwidth fairly. In Autograph, we seek to allow monitors to announce to others the `(IP-addr, dst-port)` pairs they have observed port scanning themselves, to limit the total bandwidth of announcements sent to the multicast group within a pre-determined cap, and to allocate announcement bandwidth relatively fairly among monitors. We recount the salient features of RTCP briefly:

- A population of senders all joins the same multicast group. Each is configured to respect the same total bandwidth limit, B, for the aggregate traffic sent to the group.
- Each sender maintains an interval value I it uses between its announcements. Transmissions are jittered uniformly at random within $[0.5, 1.5]$ times this timer value.
- Each sender stores a list of the unique source IP addresses from which it has received announcement packets. By counting these, each sender learns an estimate of the total number of senders, N. Entries in the list expire if their sources are not heard from within a timeout interval.
- Each sender computes $I = N/B$. Senders keep a running average of the sizes of all announcement packets received, and scale I according to the size of the announcement they wish to send next.
- When too many senders join in a brief period, the aggregate sending rate may exceed C. RTCP uses a *reconsideration* procedure to combat this effect, whereby senders lengthen I probabilistically.
- Senders which depart may optionally send a BYE packet in compliance with the I inter-announcement interval, to speed other senders' learning of the decrease in the total group membership.
- RTCP has been shown to scale to thousands of senders.

In the tattler protocol, each announcement a monitor makes contains between one and 100 port-scanner reports of the form `(src-IP, dst-port)`. Monitors only announce scanners they've heard *themselves*. Hearing a report from another monitor for a scanner suppresses announcement of that scanner for a *refresh interval*. After a *timeout interval*, a monitor expires a scanner entry if that scanner has not directly scanned it and no other monitor has announced that scanner. Announcement packets are sent in accordance with RTCP.

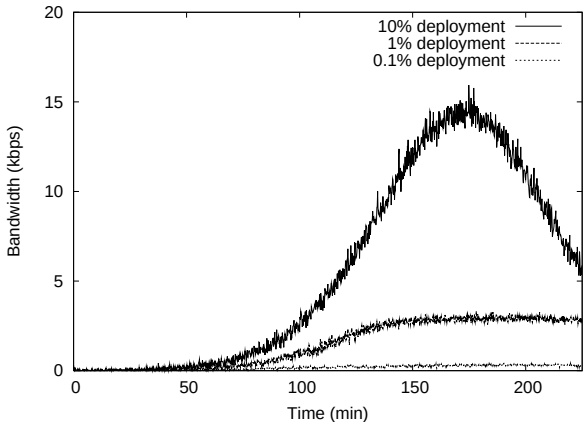


Figure 4.3: **Bandwidth consumed by tattler:** during a Code-RedI v2 epidemic, for varying numbers of deployed monitors. The peak bandwidth required by 10% deployment (630 monitors) is a mere 15Kbps.

Every time the interval I expires, a monitor sends any announcements it has accumulated that haven't been suppressed by other monitors' announcements. If the monitor has no port scans to report, it instead sends a BYE, to relinquish its share of the total report channel bandwidth to other monitors.

4.3 Bandwidth Consumption

One of the requirements in signature detection system design is the bandwidth efficiency. Now we turn to the estimation of required bandwidth to run *tattler*. We are particularly interested in the bandwidth consumption during a worm epidemic episode because the number of scanners is unusually high during the time.

Figure 4.3 shows the bandwidth consumed by the tattler protocol during a simulated Code-RedI-v2 epidemic, for three deployed monitor populations (6, 63, and 630 monitors). We use an aggregate bandwidth cap C of 512 Kbps in this simulation. Note that the peak bandwidth consumed across all deployments is a mere 15Kbps. Thus, sharing port scanner information among monitors is quite tractable. While we've not yet explicitly explored dissemination of signatures in our work thus far, we expect a similar protocol to tattler will be useful and scalable for advertising signatures, both to Autograph monitors and to other boxes that may wish to filter using Autograph-generated signatures.

Note well that "background" port scanning activities unrelated to the release of a new worm are prevalent on the Internet, and tattler must tolerate the load caused by such back-

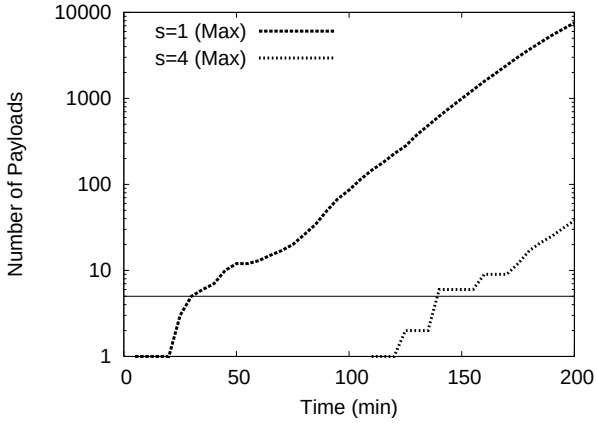


Figure 4.4: **Payloads observed over time (with Tattler):** tattler among 63 distributed monitors.

ground port scanning. `dshield.org` [41] reports daily measurements of port scanning activities, as measured by monitors that cover approximately 2^{19} IP addresses. The `dshield.org` statistics from December 2003 and January 2004 suggest that approximately 600,000 unique `(source-IP, dst-port)` pairs occur in a 24-hour period. If we conservatively double that figure, tattler would have to deliver 1.2M reports per day. A simple back-of-the-envelope calculation reveals that tattler would consume 570 bits/second to deliver that report volume, assuming one announcement packet per `(source-IP, dst-port)` pair. Thus, background port scanning as it exists in today's Internet represents insignificant load to tattler.

4.4 Worm Payload Accumulation Speed

We now measure the effect of running tattler on the time required for Autograph to accumulate worm flow payloads in a distributed deployment. Figure 4.4 shows the time required to accumulate payloads in a deployment of 63 monitors that use tattler. Note that for a port scanner detection threshold $s = 1$, the shortest time required to accumulate 5 payloads across monitors has been reduced to approximately 1500 seconds, from nearly 4000 seconds without tattler (as shown in Figure 4.2). Thus, sharing scanner address information among monitors with tattler speeds worm signature detection in this simulated worm propagation.

In sum, running a distributed population of Autograph monitors holds promise for speeding worm signature generation in two ways: it allows the "luckiest" monitor that *first* accumulates sufficient worm payloads determine the delay until signature generation, and it

allows monitors to chatter about port-scanning source addresses, and thus *all monitors* classify worm flows as suspicious earlier. Note, however, that the speed of the worm payload accumulation in each monitor is still bounded by the size and the location of each monitored network. In Chapter 5, we investigate the upper bound of the payload accumulation speed in this distributed Autograph system and propose techniques to overcome the limitation.

4.5 Online, Distributed, DMZ-Trace-Driven Evaluation

The simulation results presented thus far have quantified the time required for Autograph to accumulate worm payloads after a worm's release. We now use DMZ-trace-driven simulation on the one-day ICSI trace (that contains inbound port 80 traffic as described in Table 3.2) to measure how long it takes Autograph to identify a newly released worm *among the background noise of flows that are not worms,* but have been categorized by the flow classifier as suspicious after port scanning the monitor. We are particularly interested in the trade-off between early signature generation (sensitivity across time, in a sense) and specificity of the generated signatures. We measure the speed of signature generation by the fraction of vulnerable hosts infected when Autograph first detects the worm's signature, and the specificity of the generated signatures by counting the *number* of signatures generated that cause false positives. We introduce this latter metric for specificity because raw specificity is difficult to interpret: if a signature based on non-worm-flow content (from a misclassified innocuous flow) is generated, the number of false positives it causes depends strongly on the traffic mix at that particular site. Furthermore, an unspecific signature may be relatively straightforward to identify as such with "signature blacklists" (disallowed signatures that should not be used for filtering traffic) provided by a system operator.

We simulate an online deployment of Autograph as follows. We run a single Autograph monitor on the ICSI trace. To initialize the list of suspicious IP addresses known to the monitor, we run Bro on the *entire* 24-hour trace using all known worm signatures, and exclude worm flows from the trace. We then scan the *entire* resulting worm-free 24-hour trace for port scan activity, and record the list of port scanners detected with thresholds of $s \in \{1, 2, 4\}$. To emulate the steady-state operation of Autograph, we populate the monitor's suspicious IP address list with the *full* set of port scanners from one of these lists, so that all flows from these sources will be classified as suspicious. We can then generate a *background noise* trace, which consists of only non-worm flows from port scanners, as would be detected by a running Autograph monitor for each of $s \in \{1, 2, 4\}$. Figure 4.5 shows the quantity of non-worm noise flows in Autograph's suspicious traffic pool over the trace's full 24 hours.

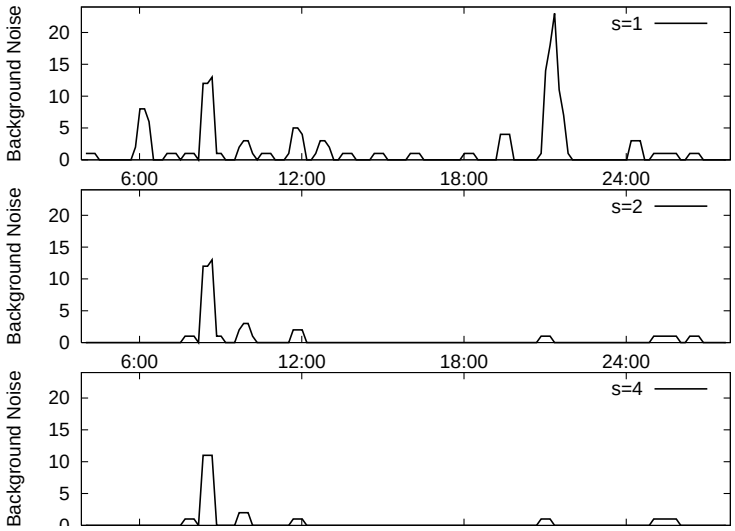
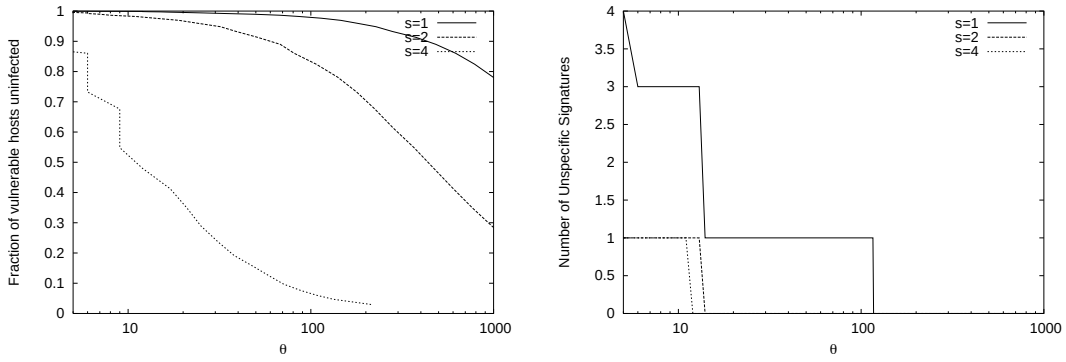


Figure 4.5: **Background "noise" flows classified as suspicious *vs.* time:** with varying port-scanner thresholds; ICSI DMZ trace.

We simulate the release of a novel worm at a time of our choosing within the 24-hour trace as follows. In this simulation, we configure Autograph to perform the content prevalence analysis every 10 minutes (while we suggest performing the analysis as frequently as possible in the real deployment, in order to generate signatures for fast propagating worms quickly), and a holding period t for the suspicious flow pool of 30 minutes. Using the simulation results from Section 4.2, we count the number of worm flows *expected* to have been accumulated by the "luckiest" monitor among the 63 deployed during each 30-minute period, at intervals of 10 minutes. Note that the luckiest monitor differs from interval to interval, and we take the count from the luckiest monitor in each interval. We then add that number of complete Code-RedI-v2 flows (available from the pristine, unfiltered trace) to the suspicious traffic pool from the corresponding 30-minute portion of the ICSI trace in order to produce a realistic mix of DMZ-trace noise and the expected volume of worm traffic (as predicted by the worm propagation simulation). In these simulations, we vary θ, the total number of flows that must be found in the suspicious traffic pool to cause signature generation to be triggered. All simulations use $w = 95\%$. Because the quantity of noise varies over time, we uniformly randomly choose the time of the worm's introduction, and take means over ten simulations.

Figure 4.6 (a) shows the fraction of the vulnerable host population that is infected when Autograph detects the newly released worm as a function of θ, for varying port scanner detection sensitivities/specificities ($s \in \{1, 2, 4\}$). Note the log-scaling of the x axis. These



(a) (b)

Figure 4.6: **Early Signature Detection vs. Quality of Signature:** (a) Fraction of vulnerable hosts uninfected when worm signature detected *vs.* θ, number of suspicious flows required to trigger signature detection. (b) Number of unspecific signatures generated *vs.* θ, number of suspicious flows required to trigger signature detection.

results demonstrate that for a very sensitive/unspecific flow classifier ($s = 1$), across a wide range of θs (between 1 and 40), Autograph generates a signature for the worm before the worm spreads to even 1% of vulnerable hosts. As the flow classifier improves in specificity but becomes less sensitive ($s = \{2, 4\}$), Autograph's generation of the worm's signature is delayed, as expected.

Figure 4.6 (b) shows the number of unspecific (false-positive-inducing) signatures generated by Autograph, as a function of θ, for different sensitivities/specificities of flow classifier. The goal, of course, is for the system to generate zero unspecific signatures, but to generate a worm signature before the worm spreads too far. Our results show that for $s = 2$ and $\theta = 15$, Autograph generates signatures that cause no false positives, yet generates the signature for the novel worm before 2% of vulnerable hosts become infected. Our point is *not* to argue for these particular parameter values, but rather to show that there exists a region of operation where the system meets our stated design goals. More importantly, though, these results show that an improved flow classifier improves Autograph—as flow classifiers benefit from further research and improve, Autograph can adopt these improvements to offer faster worm signature generation with lower false positive rates.

4.6 Discussion

We briefly discuss the additional benefit of distributed monitoring in a heterogeneous environment, and the limitation of our distributed Autograph.

Distributed monitoring in a heterogeneous environment. In our evaluation, we have assumed a homogeneous environment such that the vulnerable hosts are uniformly distributed across the 6378 ASes, and the fraction of reachable hosts in a network is uniform. However, the *IP address structure* - the arrangement of active addresses in the address space - is highly non-uniform [71], so is the vulnerable host distribution during a worm propagation [85, 87, 88]. If a network has a few reachable hosts, the Autograph monitor can identify a worm's scanning activities quickly because most scanning attempts fall to unused IP addresses. However, the monitor is slow in accumulating worm payloads crucial for signature generation. For example, *network telescopes* [86, 101] – unallocated portions of the IP address space - provide powerful tools for gathering worm-infected host IP addresses and modeling worm propagation [85, 127, 87, 88]; they do not help acquire worm payload samples if a worm sends its exploit payloads only to existent servers. In contrast, if a network has many reachable hosts, the monitor cannot easily identify suspicious addresses. Distributed port scanner detection allows Autograph to generate signatures even in such a heterogeneous environment. The Autograph monitors in the densely populated area can use the scanner lists discovered by the other monitors in sparsely populated networks.

Attacks. *Fraudulent scanner IP address reports* from unauthorized monitors cause a remote source's traffic to be included by all the distributed Autograph in signature generation. Only authorized Autograph monitors must be allowed to participate in the tattler mechanism to prevent it. There are a number of available secure group communication protocols [1, 63, 109] and group signature schemes [123, 10, 11]. Even with those authorization and group signature schemes, compromised Autotraph monitors can be used to mount the fraudulent report attack. Requiring multiple monitors to report the same scanner can make the tattler more robust against such an attack, but will slow the suspicious address identification. Similarly, requiring multiple monitors to generate payload signatures that match the same set of flows may prevent fraudulent signatures from being widely deployed, but will slow the signature generation. In Part II, we present a couple of mechanisms that utilize multiple monitors, which reduces the probability of generating fraudulent signatures when compromised monitors are present. *Source spoofed port scans* can be used to mount another type of Denial-of-Service (DoS) attacks specific to Autograph: the tattler mechanism must carry report traffic proportional to the number of port scanners. An attacker could attempt to saturate tattler's bandwidth limit with spoofed scanner source addresses, and thus render tattler useless in disseminating addresses of *true* port scanners. We discussed a simple mechanism that addresses this attack in Section 3.6.

Limitations. The distributed Autograph achieves early signature generation by improving the performance of its suspicious flow selection stage. When a worm flow enters the network, the Autograph can classify the flow to be suspicious if the source IP address is in the list of identified scanners. However, if the monitored network is small or has only a few reachable hosts, the actual number of worm payloads that enter the network is low. In our simulations of the distributed Autograph, we observed that the signature was generated from one of the biggest networks. If such large networks (/16s) did not participate in our Autograph network, that could prolong the time till the signature detection even in the presence of tattler. There are a few ways to speed up the payload accumulation. One approach is to use honeypots [58, 125] and trick worm-infected hosts to send the exploit payloads. Another approach is to share the discovered suspicious flow payloads as well as the scanner lists. Even though the number of payloads that an individual monitor receives is small, the aggregated number across multiple monitors would be large enough for payload extraction. We discuss this approach in Part II.

4.7 Summary

In this chapter, we have extended Autograph to use distributed monitoring. Our simulations of the propagation of a Code-RedI-v2-like worm demonstrate that by tattling to one another about port scanners they overhear, distributed Autograph monitors can detect worms earlier than isolated, individual Autograph monitors, and that the bandwidth required to achieve this sharing of state is minimal. DMZ-trace-driven simulations of the introduction of a novel worm show that a distributed deployment of 63 Autograph monitors, despite using a naive flow classifier to identify suspicious traffic, can detect a newly released Code-RedI-v2-like worm's signature before 2% of the vulnerable host population becomes infected. Our collected results illuminate the inherent tension between early generation of a worm's signature and generation of specific signatures.

Part II

Privacy-Preserving Signature Generation

Chapter 5

Introduction

The signature generation process requires multiple samples of worm flows, and the speed of worm payload accumulation is an important factor that determines the signature generation speed. Worm payload accumulation speed of a single monitor is determined by the size and the location of a monitored network. Like many other network security applications [146, 27, 148], the collaboration of distributed monitors helps worm signature generation systems to overcome the limitation of a single monitor.

In Chapter 4, we proposed a technique to expedite signature generation by sharing port-scanner address information among distributed monitors. In this part, we explore sharing information beyond port scanners' source IP addresses. In particular, we address the problem of sharing payload pattern strings found in distributed networks. Payload sharing across distributed monitors introduces a new privacy issue to automated signature generation systems. In this chapter, we discuss the benefit of distributed payload sharing, and the related privacy issue.

5.1 Motivation for Distributed Payload Sharing

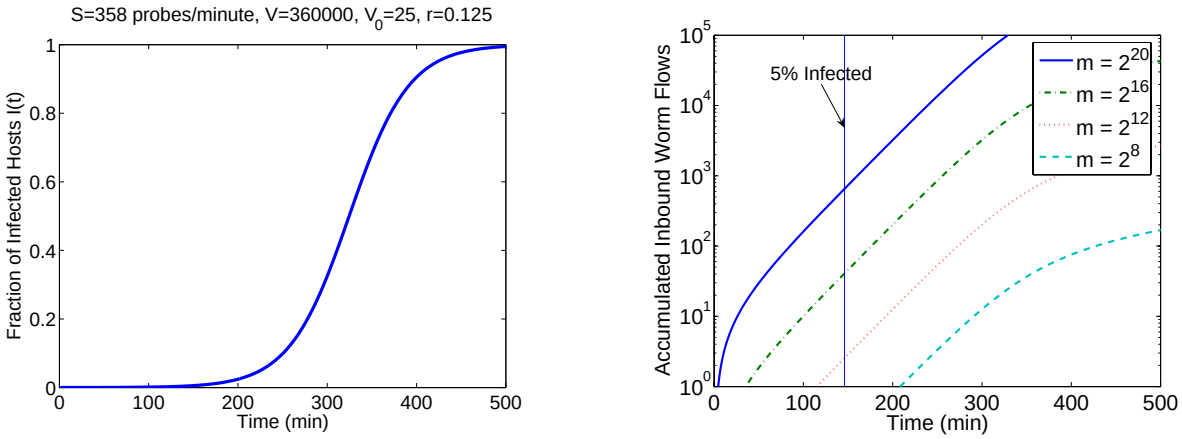
The tattler mechanism, presented in Section 4.2, helps each individual monitor to classify inbound worm flows with high probability. However, the monitor cannot accumulate more than the actual number of worm payloads crossing the boundary of the monitored network.

We first present the limitation of a single monitor by showing its worm payload accumulation speed. Our analysis shows that worm payload accumulation speed is linear to the monitored network size, the number of IP addresses that would receive worm's exploit payloads, and the total number of vulnerable hosts in the Internet. For localized scanning

worms, monitor location is also important. If a monitored network is small or far from infected host population, the monitor would not be able to accumulate enough worm payloads until a significant fraction of vulnerable hosts are infected. We discuss distributed monitoring as one of the practical solutions to overcome the limitation of a single monitor. In this work, we consider a random scanning worm and a localized scanning worm.

5.1.1 Random Scanning Worm Propagation Model

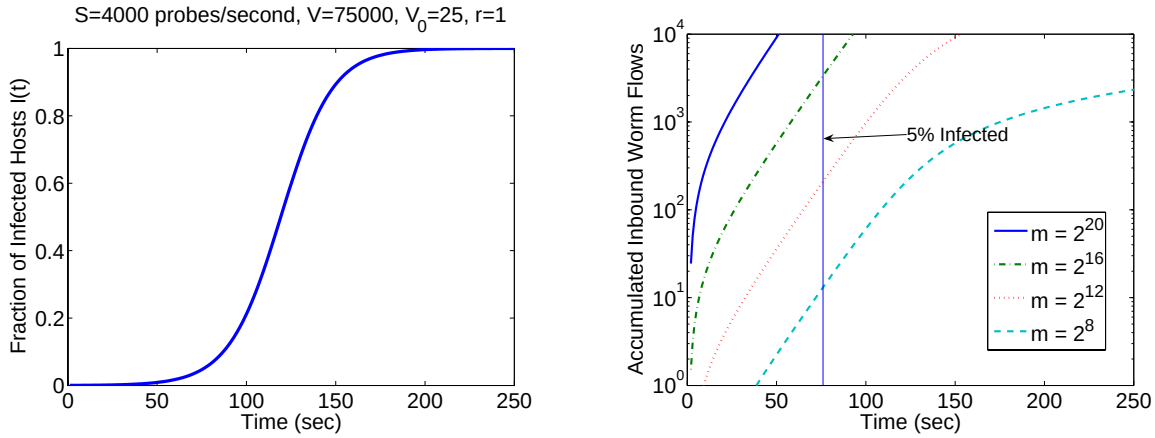
A random scanning worm, denoted by RS, choose targets randomly. The number of inbound worm flows that a monitor, installed in front of a network with m IP addresses, is proportional to the number of scans hitting the network and the number of IP addresses that would receive the worm payloads with vulnerability exploit codes. Let r be the fraction of those IP addresses in the network. If a worm requires a connection setup before transferring its exploit payload as in the Code-RedI worm epidemic [85], r is the fraction of hosts running a service on port 80. For UDP-based scanning worms such as Slammer [87], the worm sends its exploit payloads to all IP addresses, so r is 1.



(a) Simulated Code-RedI-v2 Worm Propagation (b) Received Worm Flows

Figure 5.1: **Code-RedI-v2: Number of Received Worm Payloads vs. Network Size.** (a) Simulated Code-RedI-v2 propagation with scanning rate S=358 probes/minute. (b) The actual number of received worm payloads depends on the size m of the monitored network.

We count the number of worm payloads received by a single monitor by simulating the Code-RedI-v2 and Slammer worms. We vary the monitored network size $m \in \{2^8, 2^{12}, 2^{16}, 2^{20}\}$. In this simulation, we assume 50% of the address space within the monitored network is populated with reachable hosts, and 25% of these reachable hosts run web servers. Thus, for the simulated Code-RedI-v2 worm epidemic, the fraction of worm payloads to the total



(a) Simulated Slammer Worm Propagation (b) Received Worm Flows

Figure 5.2: **Slammer: Number of Received Worm Payloads vs. Network Size.** (a) Simulated Slammer propagation with scanning rate S=4000 probes/minute. (b) The actual number of received worm payloads depends on the size m of the monitored network.

number of probes is $r = 0.5 * 0.25 = 0.125$. If there are fewer hosts running web servers in the network, the parameter r becomes smaller, so the total number of inbound worm flows would be smaller. The Slammer worm propagates with UDP and sends the worm payloads to all the scanned IP addresses. Thus, we use $r = 1$ for the Slammer simulation. We assume that none of the webservers in our monitored network is vulnerable.

Graphs in Figure 5.1 show the infection rates and the actual number of worm payloads flowing into a network since the introduction of the simulated Code-RedI-v2. The simulated worm propagates using a probe rate of $S = 358$ probes/minute [148], and the number of initial vulnerable hosts $V_0 = 25$. The total number of vulnerable hosts is $V = 360,000$ in this simulation. As shown in Figure 5.1 (a), the simulated worm infects 5% of vulnerable hosts within 2.5 hours and starts its exponential growth until most vulnerable hosts are infected. We use $I(t)$ to indicate the fraction of infected hosts at time t.

Figure 5.1 (b) shows the actual number of worm payloads coming into the monitored network during a simulated Slammer worm's propagation. This result can be considered the upper bound of the number of worm payloads in an Autograph monitor even with a perfect suspicious flow classification heuristic. The number of accumulated inbound worm payloads grows linearly with the number of infected hosts. The number is also linear to the monitored IP address space. For example, when $m = 2^{16}$, the network would accumulate 1000 worm payloads potentially, [1] before the fraction of infected hosts reaches $I(t) = 0.05$.

[1]EarlyBird [119] requires the worm sends multiple copies of its payloads during a short time period to trigger the address dispersion measurement. The graph shows the total number of payload patterns accumu-

This number is probably large enough to generate a signature. However, the monitor in a small network with $m = 2^{12}$ IP addresses, does not receive more than 3 payloads until $I(t) = 0.5$ so the monitor cannot generate a signature for the worm.

Graphs in Figure 5.2 show the simulation results for Slammer worm propagation. The Slammer worm scans the IP address space using a probe rate of 4000 probes/second [87]. [2] Such a high probe rate allows the Slammer worm to infect the most of vulnerable hosts within 200 seconds as shown in Figure 5.2 (a). This is too fast for human to react. Fortunately, the Slammer worm includes its exploit code in every probe packet. As a result, even when the probe is sent to an IP address where there is no reachable webserver, automated worm signature generation systems can use the probe payloads for their signature generation.

Figure 5.2 (b) shows the actual number of inbound worm payloads. The number grows linearly with the size of the monitored network, and the number of infected hosts. Unlike in the Code-RedI-v2 simulation, the monitor in small network receives enough number of payloads to trigger signature generation before 5% of hosts are infected. We expect that the more aggressively a worm propagates, the faster automated signature generation systems generate a signature of the worm, because the signature generation speed is linear to the payload accumulation speed. Signature detection for slowly propagating worm is more challenging in payload-extraction-based signature generation systems.

5.1.2 Localized Scanning Worms

Many worms such as Code-Red-II and Nimda employ more efficient scanning mechanisms that preferentially search for vulnerable hosts in the local address space. Consider a simplified localized scanning worm, denoted by LS, that scans the Internet as follows:

- q ($0 \leq q \leq 1$) of the scanning attempts target addresses with the same first l bits.
- $1 - q$ of the scanning attempts target randomly chosen addresses.

The average number of probes hitting a monitored network consists of two parts: (1) probes from infected hosts with the same $/l$ prefix, and (2) probes from all infected hosts in the Internet. Unlike random scanning worms, the number of worm flows caught by a monitor varies significantly based on the location of the monitored network. It is difficult

lated for a longer period of time. Thus, the graph gives a lower bound estimation on the detection time in EarlyBird-like systems.

[2] Slammer's probe is bandwidth-limited. Some infected hosts scans more aggressively. This is the average scanning rate reported in [87].

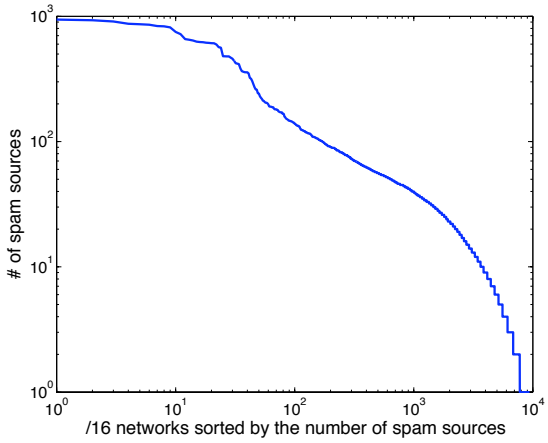
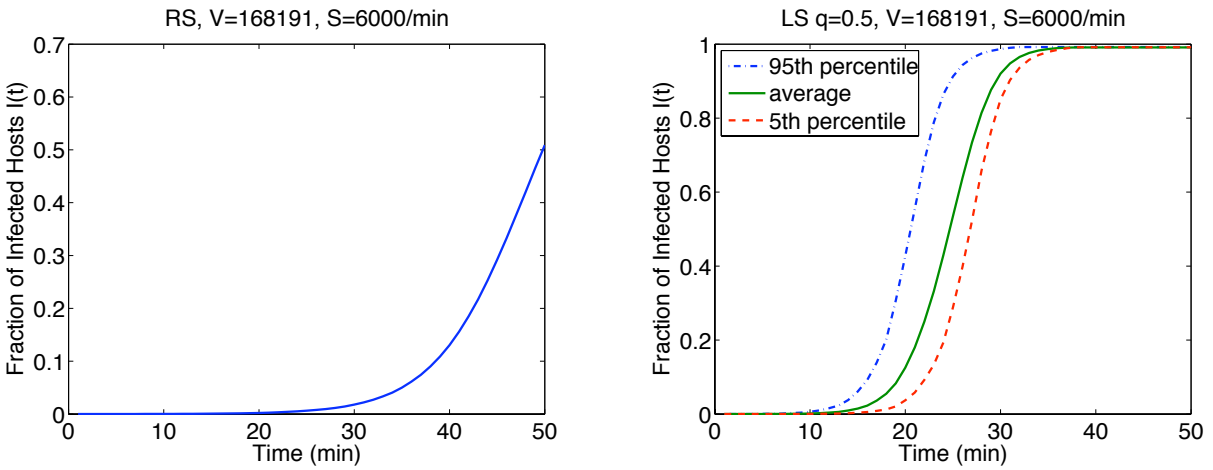


Figure 5.3: **Spam source IP distribution:** is highly non-uniform. Only 9387 /16 networks involved in the identified spam activities. This non-uniformity in vulnerable host distribution makes localized scanning worms propagate faster than random scanning worms.

to analyze the number of worm flows without considering the vulnerability distribution in the Internet. The IP address structure is non-uniform [71]; so is the vulnerable host distribution [85, 87, 88]. In order to understand the effect of monitored network size on the number of caught worm payloads, we simulate LS worm propagation using a list of e-mail spam source IP addresses. These days spammers take advantage of compromised hosts to send spam e-mails since many network service providers prevent spamming activities using their service. Thus, using spam sources as compromised hosts is not an unreasonable assumption.

We observed that 168,191 IP addresses sent spam e-mails to a spam e-mail honeypot [110] during a week of January 2006. Figure 5.3 presents the number of spam sources in each /16 prefix in a log-log plot. In log-log plots, relatively straight lines indicate that the distributions of vulnerable hosts among prefixes are far from uniform. The highly non-uniformity of (vulnerable) Internet hosts makes localized scanning more effective for fast worm propagation.

Assuming that those 168,191 IP addresses are vulnerable hosts' addresses, we simulated the propagation of LS and RS works. Figure 5.4(a) shows the 95th percentile in the result of 100 simulations of RS propagation with scanning rate $S = 6000$/minute. Figure 5.4(b) plots the results of 100 simulations of the propagation of LS with the same scanning rate and the local scanning preference $q = 0.5$. LS infects most of the vulnerable hosts within 30 minutes while RS takes 50 minutes to infect 50% of vulnerable hosts. Since the variability in the infection rate is significant, we use the 95th percentile results in the remainder of this



(a) RS (S=6000/minute) (b) LS q=0.5 (S=6000/minute)

Figure 5.4: **Scanning Strategies vs. Propagation Speed:** the non-uniformity of vulnerable host distribution makes LS more effective. With a high local scanning preference value q, LS can infect most of the vulnerable hosts in the same $/l$ network immediately after a host is infected.

work.

Assuming the vulnerable host distribution and the 95th percentile infection process acquired from the previous simulation of LS worm propagation, we count the number of worm payloads by placing a monitor in a randomly chosen network varying the monitored network size m. As in our random worm simulations, we assume that there is no vulnerable host in the monitored network but there could be vulnerable hosts with the same prefix l bits. We use $r = 1$, as in the Slammer worm propagation.

Figure 5.5 shows the actual number of worm payloads crossing the monitoring point until 5% of vulnerable hosts are infected. Notice the large variation in the number of received worm payloads. A monitor located close to a large population of infected hosts receives a huge number of worm payloads due to the localized scanning. A network located far from the initially infected host population receives as few worm payloads as in random scanning worm propagation with a probe rate of $1000 * (1 - q)$ probes/minute. The dotted line represents the number of payloads in the luckiest monitor case where our monitor is installed close to the initially infected host population. As expected, if the monitored network size is larger than the worm's local scanning space, the monitor cannot see local scanning activities, so the total number of payloads is small. The solid line represents the median case out of 100 simulations. In this case, there is no infected host with the same $/l$ prefix yet. Most of the payloads it collects are from external hosts. Thus, the location of monitors is important

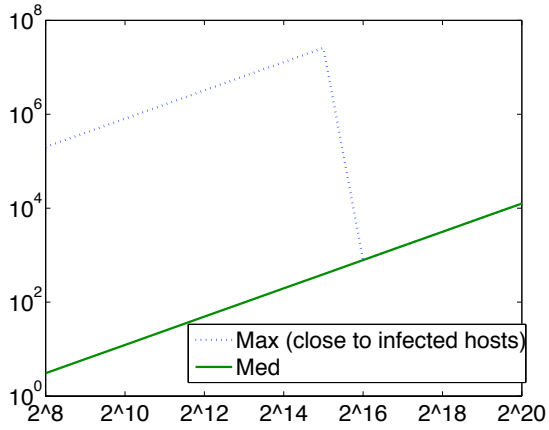


Figure 5.5: **Monitored network size vs. Number of worm payloads:** the number of accumulated worm payloads when 5% of vulnerable hosts are infected by LS (S=6000/min, q=0.5, l=16). The monitor (MAX) close to infected hosts may accumulate more than 10^5 payloads.

in signature detection for LS.

With the assumptions that all probes contain exploit payloads, and $r = 1$, a monitor with $m = 2^{12}$ could receive more than 50 worm payloads. However, if the worm propagates similarly to Code-Red-II and sends payloads only to available web servers, the total number of worm payloads a small monitor receives will be much smaller. Thus, a large monitor with many available servers is preferred for fast signature generation. The ideal case would be to place monitors close to vulnerable hosts in networks, so that monitors can capture the local scanning activities. However, without knowing a future worm's scanning strategy, placing monitors in larger networks is the safer choice.

5.1.3 Why Distributed?

For both LS and RS worm signature generation, a monitor needs to analyze the traffic of at least multiple 2^{16} IP addresses. However, monitoring such a large space presents challenges:

- Many organizations do not have such a big IP address space.
- The volume of traffic observed in such a big IP address space may be too large for a payload-extraction-based signature generation system to deal with if the IP address space is used by legitimate Internet users.

A network telescope [86] with honeypots [125] is a useful tool to enable large IP address space monitoring. We imagine that automated signature generation systems are installed

in the network telescope for fast signature generation of random scanning worms. Because the network telescope is not used by legitimate Internet users, the volume of traffic may be significantly smaller, which makes traffic analysis much easier. Even when such a large IP address space is monitored, however, if the network is not close to the initially infected host population, the chance of catching an LS worm payload may be small. If attackers know the location of a large telescope or, at least, know the space is not often by many Internet users, they can design a worm to avoid scanning the address space. Hence, having many monitors in various locations is important to generate signatures for scanning worms early.

In this part, we propose a distributed signature generation system that enables collaboration of many moderately-sized networks dispersed in the Internet. Once the monitors have individually identified a possible attack or offender, each network monitor can gain confidence in the maliciousness of the identified attacks through comparison of its findings with the observations of other monitors. A worm signature that is flagged as being possibly malicious by many monitors is more likely to be a truly malicious worm payload signature. This extra degree of confidence may also prevent unspecific signatures from being published or used to establish misguided network filtering.

5.2 Privacy Consideration

Care must be taken in the collaboration among distributed monitors. Pattern-extraction-based signature generation algorithms require the access to the network traffic payload that may contain private or proprietary information. If the private information in payloads are leaked in the payload sharing process, it can negate the benefits of improved network security. We address this privacy problem assuming that the payload content that appears in many distributed networks should not be private and can be shared as long as the source of the payload content is not disclosed.

Then our goal is to design a system that identifies and publishes a suspicious payload flagged by more than λ_G monitors. For the convenience, we call this payload a *hot item*. *Cold items*, that are not related to global anomaly, so less than λ_G monitors report, must not be disclosed because they are akin to be private or specific to a single monitor. In Chapter 7 and 8, we introduce two different techniques to protect privacy in different security models.

In order to avoid attacks from the outside of the system, we require all the participating monitors to be approved by a centralized authority and given credentials. By allowing only approved monitors to participate, we maintain the friendly operational environment resilient to attacks such as Sybil attacks [40] that use multiple fake identities, too.

Chapter 6

A Framework for Distributed Payload Sharing

Motivated by the analysis of payload accumulation speed in a single monitor, we propose a distributed payload sharing framework in this chapter. When a global worm starts its propagation, the worm payloads will be observed in many distributed networks. In contrast, we expect most innocuous flows are private, so appear in a small number of networks. Thus, if a payload pattern string appears in many places, it is likely to belong to a global worm activity.

We propose and analyze two different distributed signature detection frameworks: 1) GSD (Globally Suspicious payload pattern Detection), and 2) LSD (Locally Suspicious payload Detection). GSD is suitable when a worm scans monitored IP address space uniformly and slowly. LSD is suitable when a worm scans monitored networks in a non-uniform way as in LS.

6.1 Distributed Payload Sharing Framework

A distributed signature generation system finds wide-spreading suspicious payloads across multiple monitors and generates signatures based on the payloads. The problem consists of two parts: (a) suspicious payload identification and (b) detected signature publication. In the suspicious payload identification step, the system determines a set of payload patterns that appear in many monitoring points. In the signature publication step, local monitors notify other monitors of a new signature found locally or from the result of preceding suspicious payload identification step.

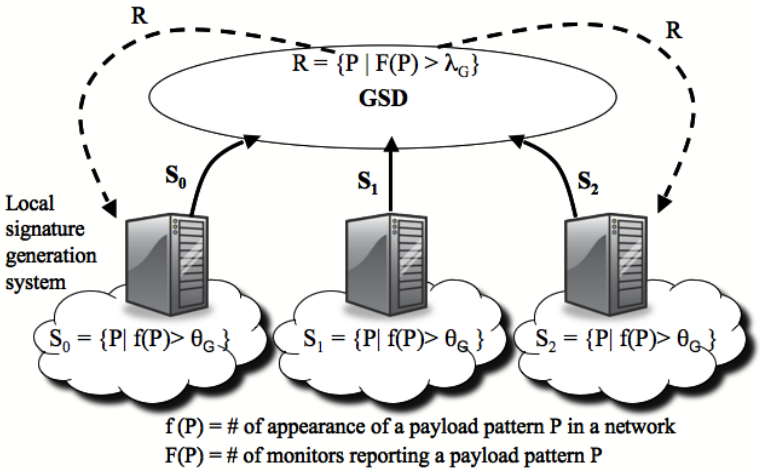


Figure 6.1: **GSD Framework Overview**

A naïve approach to the suspicious payload identification problem is to let each monitor report all the observed payloads to a trusted third party or all other monitors. The aggregate volume of traffic reported from all the monitors is tremendous, but only a small fraction of the traffic belongs to a worm's activity in the early stage of the worm propagation.

Our approach to the problem is to have each monitor perform local content prevalence analysis before distributed suspicious payload sharing. Only payload pattern strings that appear to be suspicious, based on local content prevalence analysis, are shared across monitors. Our system performs payload sharing in two different ways:

- **GSD (Globally Suspicious payload pattern Detection):** As shown in Figure 6.1, each monitor shares only a payload pattern string P whose prevalence measurement $f(P)$ exceeds a threshold θ_G .[1] When P is reported by more than λ_G monitors ($f(P) > \lambda_G$), the pattern P is likely to be a wide-spreading worm payload. The result of the distributed payload sharing process is a set of such globally suspicious payload patterns, which may look innocuous to an isolated monitor. The result R is distributed to all the monitors again. The monitors that hold the suspicious payload patterns that are in R announce their findings to other monitors if necessary.[2] The detail of GSD implementation depends on the privacy protection requirements and the number of participating monitors. We present the details of the implementation in Chapter 7 and Chapter

[1] In our implementation, $f(P)$ is a function of the number of flows that contain the payload pattern P, and the number of hosts sent the payload pattern P.

[2] The result R could be hash values or encrypted values of payload patterns due to the artifact of privacy-preserving payload sharing process. In order to generate signatures in a form usable by current content-based filtering equipments, the actual payloads should be distributed.

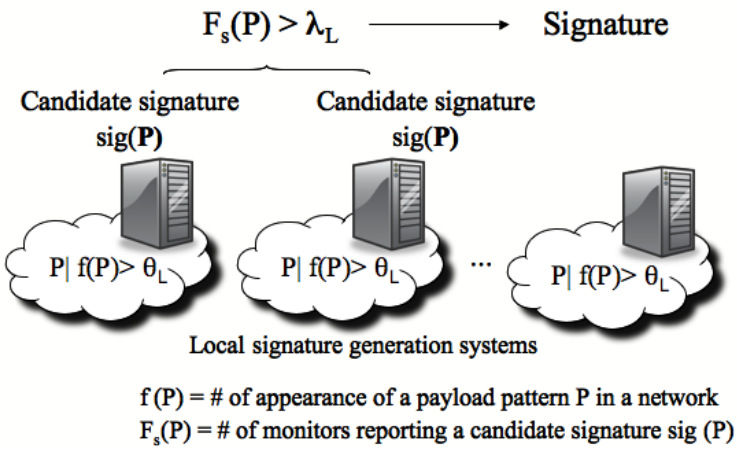


Figure 6.2: **LSD Framework Overview**

8 after clarifying our privacy protection policy.

- **LSD (Locally Suspicious candidate signature Detection):** While GSD targets to quickly catch worms that scan networks randomly, LSD targets to catch the worms that employ localized scanning techniques. As shown in Figure 6.2, each monitor reports only a payload pattern string P whose prevalence measurement $f(P)$ exceeds a threshold θ_L ($\theta_L > \theta_G$). The threshold θ_L is high enough for a monitor to generate a signature based on locally observed payload samples, so we call the payload pattern string P a *candidate signature*. If P is reported by more than λ_L ($\lambda_L \leq \lambda_G$) monitors, the candidate signature is accepted as a valid signature globally. We use $F_s(P)$ to denote the number of reports for a candidate signature based on P. We want to provide a secure mechanism that allows each monitor to report candidate signatures without revealing its identity, because some networks may not want to reveal the fact that they are under attack. We describe the details of the LSD implementation in Chapter 7.

GSD is designed to detect worms that scan the monitored networks randomly. Because random scanning worms contact each network uniformly and randomly, the number of worm flows each individual network observes would be too small to trigger local signature generation. However, the aggregate number of worm flows could be large enough to generate a signature.

LSD is appropriate when worms propagate with localized scanning and some monitors are close to the initially infected population. The monitors close to infected hosts receive enough worm flows to generate a signature locally. Then, the monitor chooses to verify its

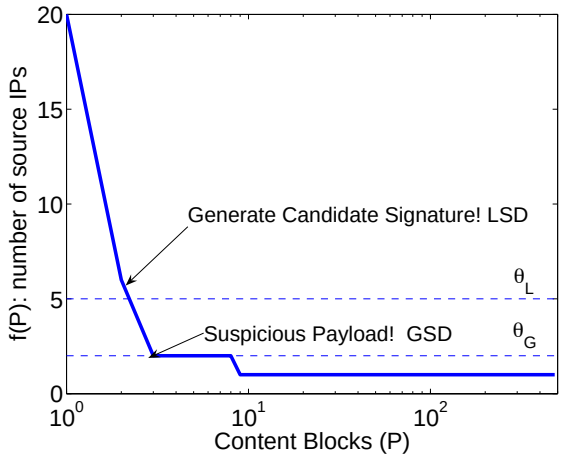


Figure 6.3: **Content Blocks in Suspicious Flow Pool and Payload Sharing Thresholds:** Prevalence histogram generated from the suspicious flow pool for port 9493 traffic. For an hour, hundreds content blocks were generated, but only content blocks whose $f(P)$ exceed θ_G are shared in GSD. If $f(P)$ exceeds λ_G , P is reported as a candidate signature in LSD. Only a handful number of content blocks pass the threshold conditions.

finding against other monitors, and distributes the discovered candidate signature to other monitors.

By applying thresholds θ_G and θ_L , we can filter out most payload patterns that do not belong to wide-spreading worm traffic. These thresholds need to be chosen based on the analysis of traffic data. We use the same traces used in the analysis in Section 3.4.1. Figure 6.3 shows the prevalence histogram of content blocks accumulated in the suspicious flow pool of an Autograph monitor in front of one of the /19 networks in the campus network. The y-axis is the frequency of each content block. Only a small fraction of the content blocks will be shared in GSD if we use $\theta_G = 2$. A much smaller fraction of the content blocks that appear more than $\theta_L = 5$ will be announced through LSD as candidate signatures.

If θ_G and θ_L are too low, the effect of filtering becomes minimal, and almost all content blocks that each monitor generates should be reported to the distributed payload sharing infrastructure. There is also a trade-off between the threshold values, and the probability of signature detection. If the threshold for filtering, θ_G and θ_L is too high, even worm payloads cannot pass the local prevalence testing in the early worm epidemic, which reduces the benefit from distributed monitoring. Similarly, we gain confidence in the result generated with large λ_G and λ_L values but large λ_G and λ_L slow the detection speed.

6.1.1 Honeypot

The number of worm payloads a monitor captures is also proportional to the fraction r of worm scan payloads containing the exploits. In most recent UDP worm epidemics, we observed the worms' scanning payloads also contain the exploit payloads; a vulnerable host is infected as soon as a scanning payload reaches the host. Accumulating enough payloads of such UDP worms may not be difficult because $r = 1$. It is more challenging to generate signatures for worms that propagate over TCP connections or send exploits only to the vulnerable hosts. In these cases, r is small.

One way to boost r even for the TCP worms is to use honeypots [125, 134]. Using honeypots is a well-known technology in worm detection and used by various network attack detection systems [105, 125, 4, 36]. Network administrators can configure their networks so that all traffic to unoccupied IP addresses or unused ports is redirected to a set of honeypots. The honeypots respond to worms' connection attempts so that the worms are tricked into transferring their exploit payloads to the honeypots. By doing so, a monitor can capture all the inbound scanning activities sent to unoccupied IP addresses and their exploit payloads.

6.1.2 Analysis of GSD

For random scanning worms, the signature generation problem under this framework is analogous to the problem of allocating balls into a set of buckets; after throwing a large number of balls randomly, how many buckets will have more than θ_G balls? The probability a worm payload enters a network with m IP addresses is $m \cdot r/\Omega$ where Ω is the size of worm's scanning address space. Let E^{θ_G} denote the event where a monitor i fails to collect $\theta_G + 1$ worm payloads after a worm scans B IP addresses. Assuming the tattler mechanism is perfect and all the worm flows entering monitored networks can be classified as to be suspicious, the probability of E^{θ_G} is:

$$\begin{aligned}
\mathbb{P}[\text{monitor } i \text{ collects less than } \theta_G + 1] &= \mathbb{P}[E^{\theta_G}] \\
&= \sum_{j=0}^{j=\theta_G} \lim_{B\to\infty} \binom{B}{j} \left(\frac{mr}{\Omega}\right)^j \left(1 - \frac{mr}{\Omega}\right)^{B-j} \\
&\approx \sum_{j=0}^{j=\theta_G} \frac{\gamma^j \cdot e^{-\gamma}}{j!} \text{ where } \gamma = \frac{mrB}{\Omega}
\end{aligned}$$

From the above equation, we derive the probability that more than λ_G monitors collect more than θ_G payloads:

$$\mathbb{P}[\text{more than } \lambda_G \text{ monitors collect more than } \theta_G \text{ worm payloads}] = 1 - \sum_{i=0}^{\lambda_G} \binom{M}{i} (1 - \mathbb{P}[E^{\theta_G}])^i (\mathbb{P}[E^{\theta_G}])^{M-i}$$

The total number of scans B sent to the Internet at time t is determined by the number of infected hosts at the time. Let $N(i)$ be the number of infected hosts at time i. When the measurement time interval is w and the worm's average scanning rate is s, the total number of probes that infected hosts send to 2^{32} IP addresses is $N_{(t,w)} = \sum_{i=t-w}^{t} N(i) \cdot S$ for a random scanning worm. Assuming that infected hosts are uniformly distributed across the Internet, the number of scans that may target one of our monitored networks is $N_{(t,w)} \cdot (1 - (\frac{m}{2^{32}})^2)$.

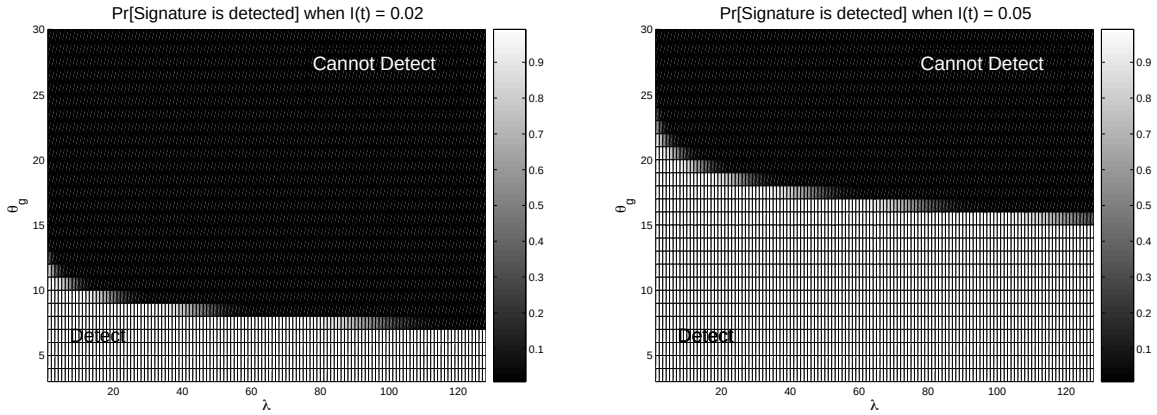
Figure 6.4 shows the probability of signature detection varying λ_G, θ_G, and the fraction of infected hosts $I(t)$ when 1024 monitors collaborate. We assume that $m = 2^{13}$, $\Omega = 2^{32}$, $r = 0.2$, and $S = 358$ scans/minute. A white cell in Figure 6.4 denotes a (λ_G, θ_G) pair for which the system can collect more than λ_G payload patterns for the worm with probability 1. Conversely, a black cell denotes a (λ_G, θ_G) pair for which the system cannot collect more than λ_G worm payload patterns by the time.

As expected, the lower θ_G and λ_G are, the earlier the system can identify worm payload patterns. Our model assumes a random scanning worm and a homogeneous network, and thus most of the monitors report the worm payload pattern even in the beginning of the worm propagation if θ_G is low.

Figure 6.5 shows the probability when $M = 128$ monitors collaborate. If we use the same λ_G as in the 1024 monitor case, the detection probability is lower than when we use 1024 monitors.

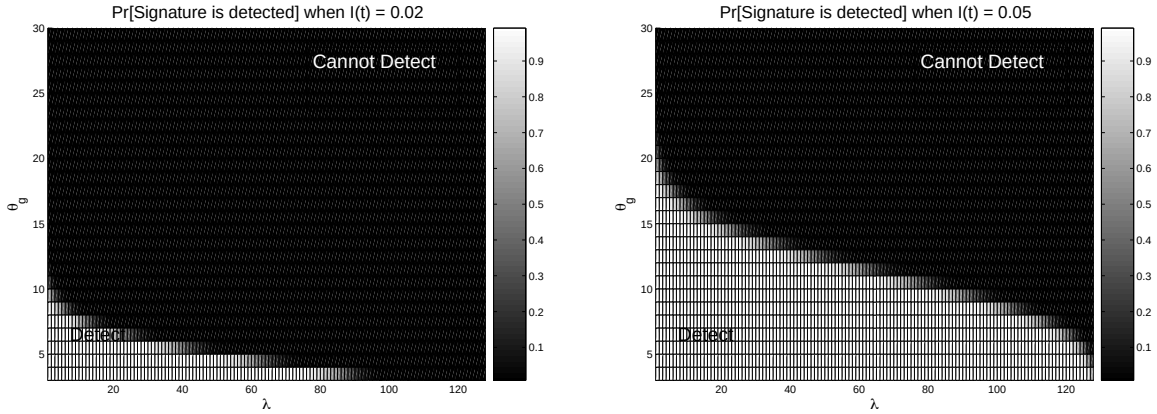
6.1.3 Analysis of LSD

Analysis of LS propagation, without considering the vulnerable host distribution and worm's scanning strategy, is difficult, and so is the analysis of LSD. To simplify the analysis, we assume that all vulnerable hosts with the same $/l$ prefix are infected immediately once an internal host is infected by LS. If each monitored network space is smaller than a $/l$ prefix network, the monitor can see many scanning activities from the infected hosts, and the num-



$I(t) = 0.02$ $I(t) = 0.05$

Figure 6.4: **1024 GSD Monitors: Signature Detection Probability vs. θ_G, λ_G and infection rate $I(t)$.** The white cells indicate that GSD finds a signature for our simulated Code-RedI-v2 worm with a probability of 1 when the combination of corresponding λ_G and θ_G is used. The signature detection probability is governed mostly by θ_G in random scanning worm signature detection.



$I(t) = 0.02$ $I(t) = 0.05$

Figure 6.5: **128 GSD Monitors: Signature Detection Probability vs. θ_G, λ_G and infection rate $I(t)$.** Since the total number of monitors is only 128, the global threshold λ_G should be selected accordingly. If we use a local threshold θ_G lower than 5 and a global threshold λ_G lower than 38, we can detect the signature of this example random worm before 2% of hosts are infected.

ber of outbound scans easily exceeds θ_L . There are M monitors that monitor m IP address space each. Then, the total number of candidate signatures reported by LSD is the number of infected $/l$ prefix networks in the Internet. Thus, the distributed system can generate a signature when λ_L monitored networks are infected.

6.2 Simulation Results

In our experiments, we simulate the detection of RS and LS worms that have a vulnerable population $V = 168,191$, and a scanning rate $S = 6000$ scans/min. For vulnerable hosts, we use the source IP addresses of spams, as in Section 5.1.2. We use the 95th percentile infection data from Section 5.1.2. The LS worm's local scanning preference parameters are $l = 16$ and $q = 0.5$. We test two different strategies for using LSD and GSD, varying the number of monitors M. Throughout this work, we evaluate the signature generation speed in terms of the fraction of infected hosts when a signature for the worm is generated, and we call it *signature detection time*.

- **Strategy I (Only GSD):** a signature is generated and deployed when more than λ_G monitors report the worm payload P during a measurement interval $\tau = 1$ hour.
- **Strategy II (Only LSD):** a signature is generated and deployed when more than λ_L monitors report the worm payload P as a candidate signature.

Each monitor analyzes both inbound and outbound traffic of a 2^{13} IP address space. The monitor has a honeypot so that it can analyze all inbound traffic to unoccupied IP addresses or unused ports ($r = 1$). For outbound traffic, it is difficult to use the honeypot. Thus, the monitor analyzes the outbound traffic successfully transferred to external hosts. We assume that a worm finds a host in the same $/l$ network with a probability of 0.16; it finds a host in the Internet with a probability of 0.02.

For comparison, we use two monitors installed in a randomly chosen 2^{16} IP address network. One monitor analyzes all inbound traffic to the network, and 12.5% of the IP addresses have the worm's target port open. Another monitor has a honeypot that responds to all connection attempts to unused IP addresses or ports. Both monitors generate a worm signature when they observe more than $\theta = 15$ payloads. The parameters were chosen according to the simulation results in Section 4.5. If multiple monitors collaborate, GSD needs some more time for correlating reported suspicious payloads and distributing detected signatures, after λ_G monitors report a worm payload. We call this delay *lag* and test the cases where the lag

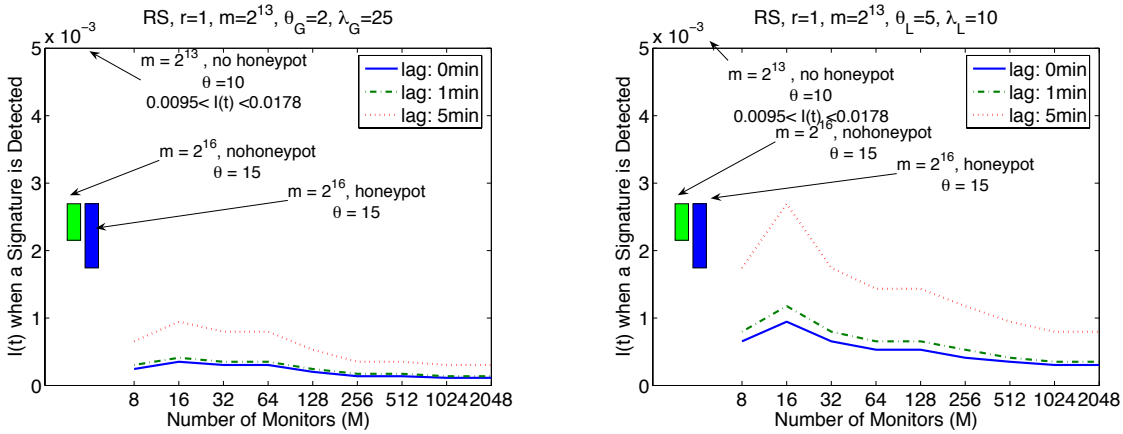
is $\{0, 1, 5\}$ minutes. In LSD, each monitor maintains a counter for each candidate signature. A candidate signature is accepted when the $\lambda_L + 1$-th report is announced. We test three cases where the announcement delay is 0, 1, and 5 minutes respectively. Experiments with a single monitor do not require any distributed operation so the detection time is exactly when $f(P)$ exceeds a threshold.

Strategy I: Only GSD. Figure 6.6 (a) and Figure 6.7 (a) show the detection time measured in the fraction of infected hosts for a random scanning worm (RS) and a localized scanning worm (LS) respectively. A payload pattern string is accepted as a signature if more than λ_G monitors report the same string. In this experiment, we set $\lambda_G = 25$. Based on the GSD analysis in Section 6.1.3, the system with this threshold should be able to generate the worm signature before 2% of vulnerable hosts are infected. The squared regions show the detection time when we monitor a single 2^{16} IP address space with the signature generation threshold $\theta = 15$.

When a worm scans randomly, worm payloads are evenly distributed across all the monitors. The detection time of GSD with 2^{13} IP address space monitoring is much lower than that of the monitor installed in 2^{16} IP address space. If a honeypot is also deployed in the 2^{16} IP address space, the monitor can detect the signature much earlier. However, multiple distributed monitors generate a signature even earlier due to the low local detection threshold θ_G . For GSD, we use a small content prevalence threshold $\theta_G = 2$. Because many monitors report the same content blocks, we can accept the payload pattern as a signature with high confidence.

For a localized scanning worm, we also observe that the larger the number of participating monitors, the faster the system can generate the worm signature. The benefit from multiple monitors is more obvious in LS signature detection than in RS detection. This is because, with a large number of monitors, we have a higher chance of having monitors close to infected hosts. Even when there is a lag between signature generation and signature distribution, GSD with more than 256 monitors could detect the signature as early as the monitor with a honeypot and a 2^{16} IP address space.

Strategy II: Only LSD. Figure 6.6 (b) and Figure 6.7 (b) show the detection time when we use LSD for RS and LS, respectively. LSD with multiple monitors must wait until more than 10 monitors report the same candidate signature. If there are fewer than 20 monitors, we require LSD to generate signatures when half the monitors report the same candidate signature. The detection time is higher than the case of 2^{16} IP address monitoring when the number of monitors is less than 32 and the worm performs localized scanning. However, if we have more than 128 monitors, the performance is similar to that of 2^{16} IP address



(a) Strategy I: Only GSD

(b) Strategy II: Only LSD

Figure 6.6: **Random Scanning Worm Detection Time:** Time is measured as the fraction of infected hosts when the detection system completes a signature identification. GSD with 128 2^{13} IP address spaces detects the signature when less than 0.1% of vulnerable hosts are infected. LSD with 128 monitors detects the signature when less than 0.2% of hosts are infected.

monitoring. This is because we could place some of our monitors close to infected hosts.

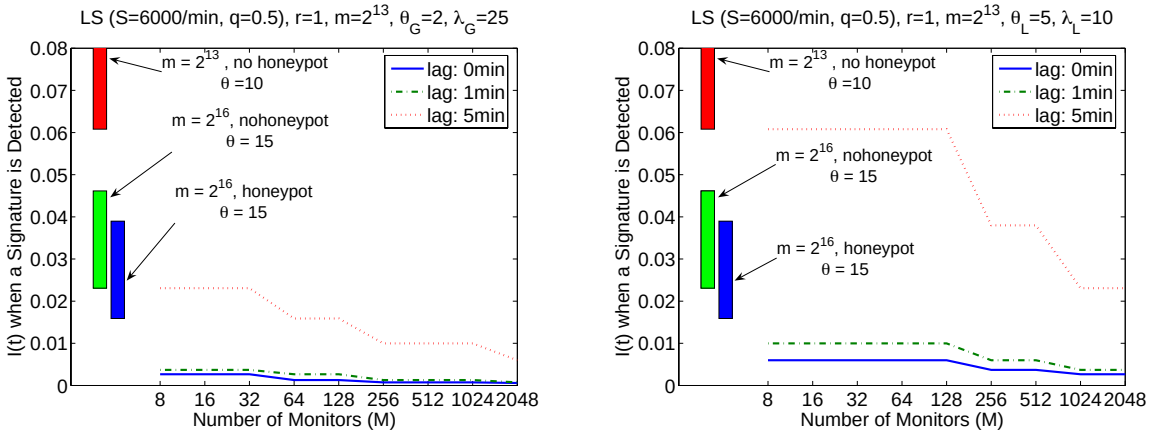
For random scanning worms, the signature detection time is higher than that of GSD because the number of payloads received is limited by the size of the monitored network. However, the LSD did detect the signature before 0.2% of hosts are infected.

In sum, GSD is good when a worm scans randomly, or the monitored networks are far from the initially infected population. LSD is good for localized scanning worm signature detection. This suggests that, by combining GSD and LSD, we can develop a system that is suitable both for both localized scanning worms and random scanning worms.

6.3 Legitimate Traffic

The distributed framework we are proposing utilizes the characteristics of wide-spreading malicious payloads. We assume the payload pattern belongs to worm traffic if it appears in many locations. However, can the system avoid misclassifying legitimate traffic as worm traffic? If it misclassifies legitimate traffic as worm traffic, will it be dangerous to publish the identified payload patterns?

To answer these questions, we study real traffic traces. The traces contain full payloads of `smtp` (port 25), `http` (port 80), `p2p` (port 9493) traffic collected at a border router of a university campus network on Aug 25, 2006 for an hour. The details of the trace are as de-



(a) Strategy I: Only GSD (b) Strategy II: Only LSD

Figure 6.7: **Localized Scanning Worm Detection Time:** Time is measured as the fraction of infected hosts when the system completes a signature identification. GSD with more than 128 2^{13} IP address space detects about 2% of hosts are infected. LSD outperforms a 2^{16} IP address space monitor when the lag is lower than 1 minute.

scribed in Section 3.5. The network consists of more than 2^{16} IP addresses that spans across departments, offices, dormitory, and wireless networks. For our experiments, we divide the network into 8 networks and simulate a distributed system consisting of 8 monitors. Each network has a contiguous 2^{13} IP address space.

All monitors perform suspicious flow classification based on the port scanner list learned from the tattler mechanism. Then, they generate content blocks from the captured suspicious flows and perform Autograph's local content-prevalence analysis method. Because the port-scanner heuristic is not perfect, there are misclassified flows in the suspicious flow pool. We use the average content block size of 64 bytes, which is larger than the block size of 16 bytes that performed best in the experiment in Section 3.4.1. We choose 64 bytes because 16 bytes average block size results in higher false positive rates for P2P application traffic. The average block size needs to be adjusted depending on the characteristics of traffic per port.

We assume that each monitor checks the number of source IP addresses that have sent the flows. Content blocks sent by less than a source IP addresses are not included in the global payload sharing process.

Figure 6.8 shows the breakdowns of content blocks from suspicious flows in port 80 (`http`) based on the number of networks $F(P)$ that have generated the same content block. The number on top of each bar indicates the total number of content blocks that are reported by the corresponding monitor. When no local threshold θ_G is applied, the monitor in `net0` reports 136 content blocks. However, only 11 of them are also found in other networks. Two

content blocks are common across $F(P) = 5$ networks. When we apply a high θ_G (Figure 6.8), we can filter most content blocks and reduce the required bandwidth for payload sharing. For example, the monitor in `net0` has only one content block that appears in other two monitors' networks.

Figure 6.9 shows the breakdowns of content blocks from port 25 (`smtp`). Suspicious flows for port 25 can be long e-mail messages that contain files and many images. Thus, each network monitor could generate thousands content blocks easily. There are many content blocks common across 6 networks. However, after filtering out the content blocks sent by only one or two source IP addresses, all monitors but the one in `net0` have an empty set of content blocks to be reported. Those 80 content blocks are from spam e-mails sent by two hosts. The emails contain long MIME-encoded content and generate many content blocks.

Figure 6.10 shows a similar trend for port 9493 (`p2p`). The networks that have P2P clients are contacted concurrently by a few mistakenly accused scanner IP addresses. All P2P file search requests through the IP addresses are captured in suspicious flow pools. So, when we use $\theta_G = 0$, about half of the reported content blocks from each monitor are common. Also, the actual number of content blocks is large because many P2P search requests are forwarded through the IP addresses. If we apply $\theta_G = 1$, each monitor needs to report only fewer than 16 content blocks. The local threshold θ_G reduces the number of content blocks significantly.

We expect, however, that the number of shared payload patterns would become larger as more networks participate in this distributed system. Higher threshold parameters would help reduce the probability of the false positives, but reduce the probability of catching worms.

We examined the payload pattern substrings reported by multiple monitors. If they are a part of private information, our distributed approach to signature generation is flawed. Fortunately, we could not find any patterns belonging to private information. We list what caused the frequent substrings to appear in many networks.

- `HTTP`: footers of HTTP messages that specify the browser types and the recognizable applications; web crawler traffic; parts of HTML structures frequently used by popular web message board software.

- `SMTP`: e-mail spams [3]. Unfortunately there are a couple of prevalent substrings that seem to belong to newsletters, sent from multiple source IP addresses. More study on the right threshold for SMTP traffic is still needed.

[3]The large number of reports is due to a long spam e-mails sent from multiple senders.

- `P2P (FileGuri)`: parts of control messages that specify the client software versions and applications, and popular queries.

Since we examined traces only from a single site and only for three protocols, it is too early for us to draw a generalized conclusion about the commonality in suspicious flows. We need further investigation with traffic from other sites and for other ports. Note, however, `smtp`, `http`, and `p2p` protocol traffic is the most frequently captured in the suspicious flow pool, and the byte volume of web, e-mail, and p2p traffic accounts for the majority of the network traffic in our experience with Autograph. The traffic for those ports contributes most of false signature generation in our experience with Autograph.

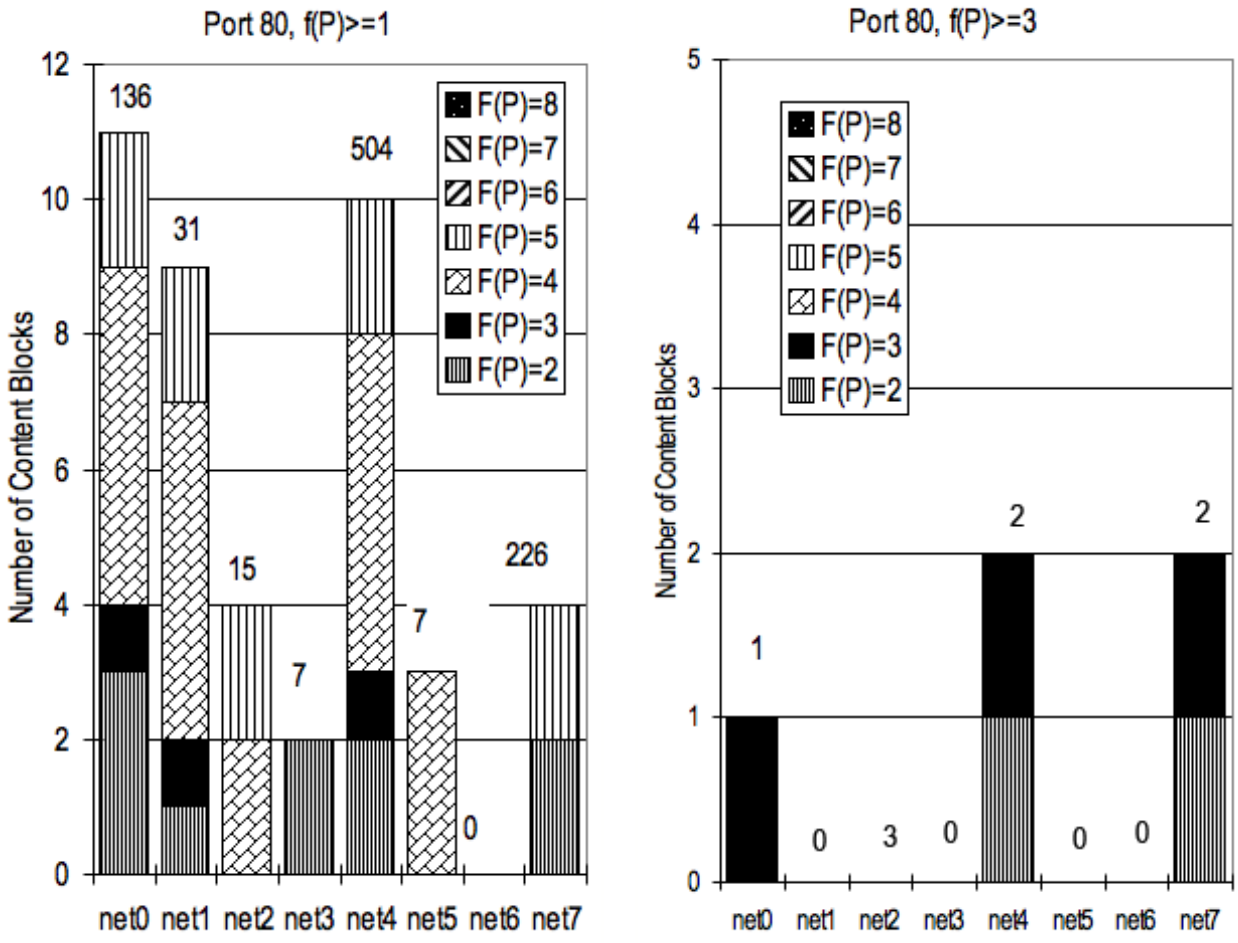
Even though the content blocks commonly observed across the monitored networks are not private, they are still problematic if we want to deploy the signatures generated by Autograph in automatic content-based filtering boxes. However, the set of such false signatures can be avoided by using a signature blacklist, and a different set of parameters depending on the traffic mix for each port. The blacklist contains the commonly used HTTP headers and footers, web crawlers' signatures, and P2P application message headers. A handful of blacklist entries would be sufficient to suppress the prevalent innocuous payload pattern strings found in our traffic trace.

6.4 Summary

The speed of pattern-extraction-based signature generation depends on how quickly a system can capture enough worm payload samples to trigger its content-prevalence analysis. We have proposed a distributed signature generation framework designed for moderately-sized networks to collaboratively generate signatures for both random scanning worms and localized scanning worms. By using distributed monitors, we can increase the chance of placing monitors close to vulnerable hosts, which is important for localized scanning worm signature generation. Distributed monitors allows us to generate signatures for random scanning worms earlier than one single monitor that monitors a large network (i.e., a 2^{16} IP address space). A prerequisite assumption under our distributed monitoring framework is that wide-spreading worm payloads are observed from many locations; legitimate private traffic payloads are not observed from many locations. We examined this assumption by studying a real traffic trace collected from a class-B IP network. From our study, we do find that some payload patterns are commonly found from multiple monitoring points but they are not private. More importantly, they could be suppressed with a short signature

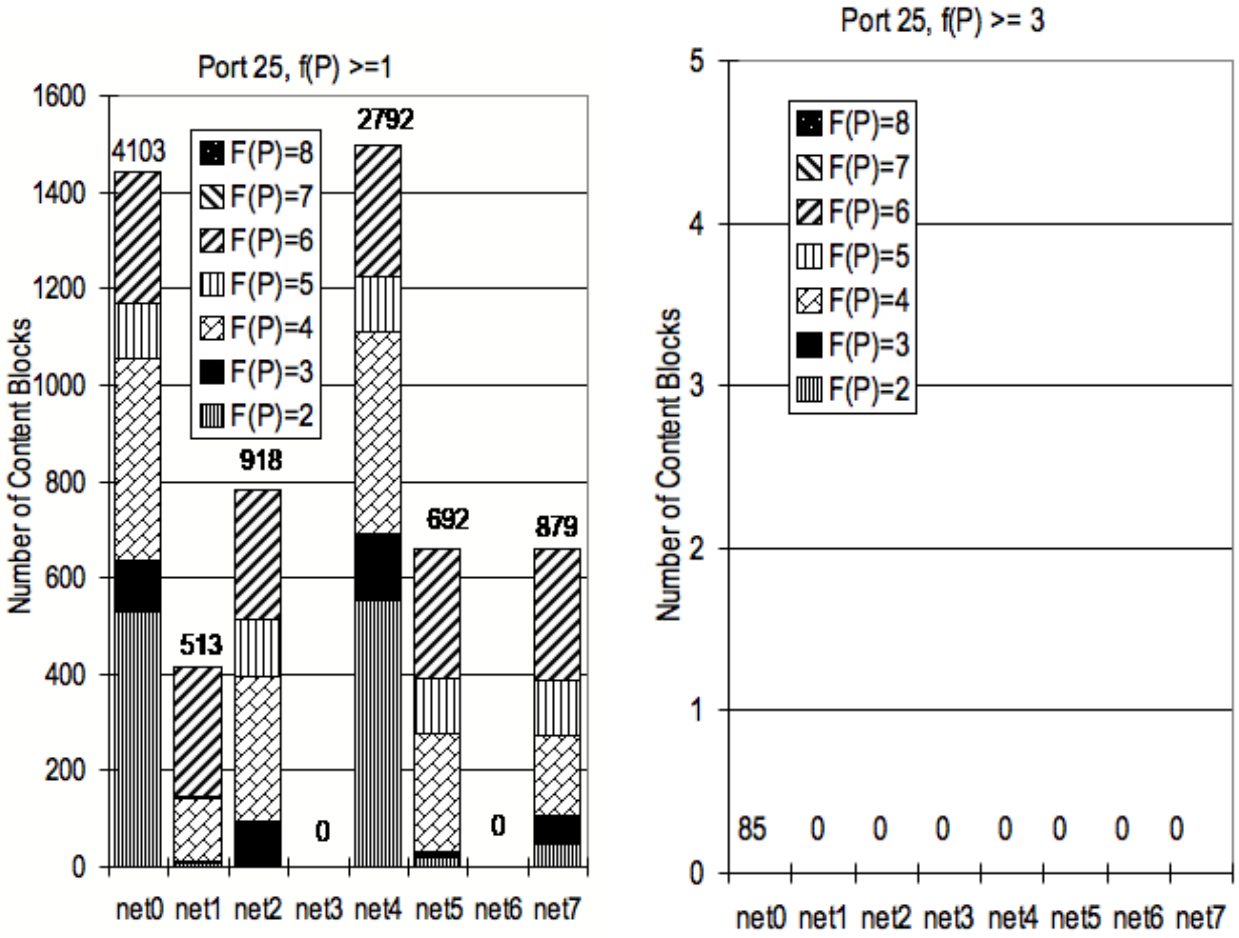
blacklists.

In the next chapters, we propose two payload sharing techniques that protect the privacy of uncommon, possibly private payload patterns under our payload sharing frameworks.



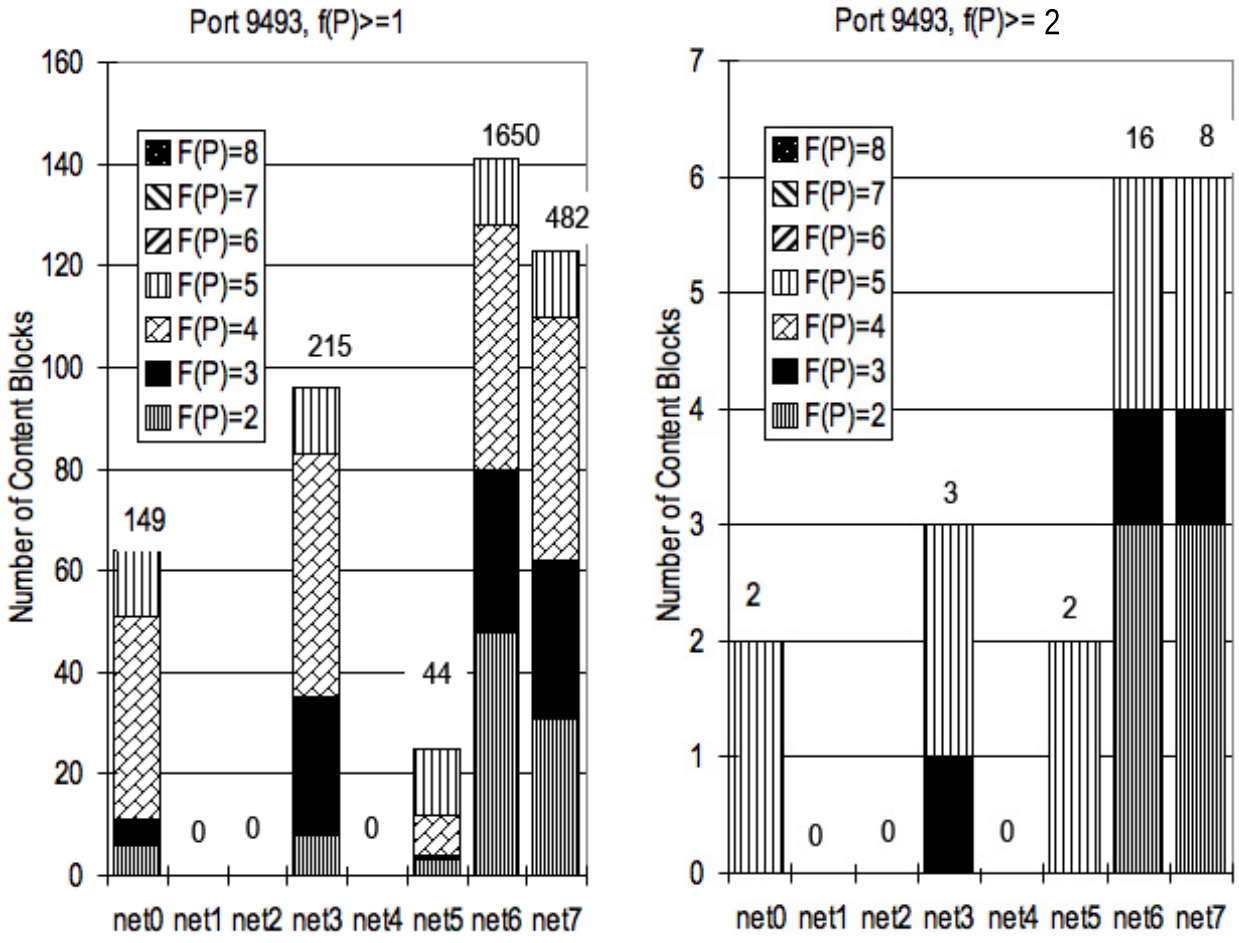
(a) $\theta_G = 0$ (b)$\theta_G = 2$

Figure 6.8: **Payload Pattern Sharing:** `HTTP` **port 80 traffic.** Breakdown of content blocks based on the number of networks reporting the same content block. Each number on top of a bar shows the total number of content blocks that are reported by the monitor. With a higher local threshold ($\theta_G = 2$), most content blocks are filtered out. For example, the monitor in net2 reported 3 content blocks in total but none of them appeared in other networks. Those content blocks.



(a) $\theta_G = 0$ (b) $\theta_G = 2$

Figure 6.9: **Payload Pattern Sharing:** `SMTP` **port 25 traffic.**



(a) $\theta_G = 0$ (b) $\theta_G = 1$

Figure 6.10: **Payload Pattern Sharing:** `p2p` **port 9493 traffic.**

Chapter 7

Preserving Privacy using HOTITEM-ID

Selecting a worm payload pattern out of a large volume of data while preserving the privacy of thousands of participating monitors is a difficult problem. In this chapter, we develop a technique to perform privacy-preserving suspicious payload sharing across thousands of distributed monitors. The technique protects the owner and data privacy of participating monitors by hiding non-popular private items in a crowd of indistinguishable elements. An adversary cannot determine which honest network produced a content block (owner privacy) or what private information is in reported content blocks (data privacy). False positives will generally be rare, and thus hidden in a large crowd.

The technique is an application of the HOTITEM-ID protocol proposed by Kissner *et al.* [69]. The protocols use approximate heavy-hitter detection technique, approximate distinct counting technique, anonymous routing, and a novel cryptographic tool. Section 7.2 briefly summarizes the HOTITEM-ID protocol. Section 7.3 presents the adoption of the protocols for privacy-preserving distributed signature generation. Section 7.4 presents our experimental results to show the efficiency of our technique. Detailed analysis of the privacy protection can be found in the original paper [69].

7.1 Overview

In our problem setting, each monitor M_i ($0 \leq i < M$) in the system holds a private dataset S_i of locally identified n content blocks. The monitors are connected by a peer-to-peer network. A *λ-threshold hot content block* is a content block P that appears in at least λ distinct monitors' private input datasets. Our goal is to find the content block P that appears in at least λ distinct monitors' private dataset. R_λ is the set of all λ-threshold hot items in the system. All

items not in R_λ are called *cold items*.

7.1.1 Adversary Model

The adversary can eavesdrop on *all* communication. We assume that a fraction c of the monitors may maliciously and arbitrarily misbehave, while the rest of the monitors are honest-but-curious.

Honest-but-curious Monitors. A honest-but-curious monitor will (1) follow the protocol as specified, and (2) not collude with other monitors to gain information, though she may try to distill information from the messages she receives. For example, a honest-but-curious monitor casually browse all the reported content blocks and attempt to find user passwords transferred in a clear text or interesting web content addresses.

Malicious Monitors. A malicious monitor might not follow the protocol; instead it might behave arbitrarily. For example, malicious monitors could collude such that if cM exceeds λ, they can make an arbitrary cold item hot by each reporting it as being in their private input set. However, this problem is out of the scope of this work. Because any monitor has the freedom to choose its private input set, any protocol in this setting is vulnerable to this manipulation. We focus on the secure payload sharing when at most cM monitors are malicious.

A malicious monitor could mount attacks on the distributed hot item identification system claiming to have many copies of one item, and try to boost the frequency of a cold item to be high enough to become a hot item; we call this the *inflation attack*. The HOTITEM-ID protocol ensures that each monitor can only contribute to the frequency count of an item at most once by using a novel cryptographic method called *one-show tags* (Section 7.2.2). Note that the *one-show tags* can be applied to address our candidate signature publication problem in LSD. We discuss the detail in Section 7.3.1.

Moreover, malicious monitors could attempt to forge cryptographic signatures, send bogus reports, try to learn honest monitors' private data, or fool other monitors. The HOTITEM-ID protocol defends against all of these attacks.

7.1.2 Correctness and Privacy Protection

The HOTITEM-ID protocol guarantees the correctness and privacy protection defined by the followings:

Correctness: Given the false positive rate δ_+ and the false negative rate δ_-:

- $\forall P \in S_i \cap R_k$, monitor M_i learns that $P \in R_k$ with probability at least $1 - \delta_-$.
- $\forall P \in S_i \cap R_k$, monitor M_i learns that $P \notin R_k$ with probability at least $1 - \delta_+$.

Owner Privacy: no coalition of at most cM malicious PPT (Probabilistic Polynomial Time) adversaries can gain more than a negligible advantage in associating an item P with a honest monitor M_i ($0 \leq i < M$) such that $P \in S_i$, because all monitors simply are given the set of hot items R_k and their frequencies.

Data Privacy: The HOTITEM-ID protocol provides probabilistic privacy protection rather than strong privacy protection discussed in many cryptography papers [70]. The degree of the privacy of an item P describes the size of the 'crowd' in which the content block is hidden. We use the size of the crowd to measure the degree of privacy in this work. A content block P which appears in F_P monitors' private input sets has $\Phi(F_P)$-degree of data privacy if no coalition of at most cM malicious probabilistic polynomial-time (PPT) adversaries can distinguish the pattern P from an expected indistinguishable set of $\Phi(F_P)$ content blocks. [1] Thus, for a cold item P, the larger $\Phi(F_P)$ is the better protected it is in general. In other words, with a large $\Phi(F_P)$, it is less likely that P is known to other participants that do not own P.

The HOTITEM-ID protocol is highly efficient in that it has constant per-monitor communication overhead. The efficiency is important for a large scale distributed privacy-preserving signature generation system because the system needs to generate signatures before a fast worm widely spreads. Even when thousands of monitors participate in the process, the HOTITEM-ID achieves its efficiency by adopting a relaxed notion of privacy instead of cryptographically guaranteed privacy. However, the relaxed notion of privacy provides sufficient protection for our purpose.

7.2 HOTITEM-ID Protocol

In Figure 7.1, we show an overview of the components of the efficient privacy-preserving hot item identification protocol HOTITEM-ID . We first introduce the intuition behind each component. The full detail of each component is described in our paper [69]. In Section 7.2.6, we describe the full construction of HOTITEM-ID. Once all monitors learn the hot items in their private datasets, they can distribute the signature using the mechanism described in Section 7.3.1.

[1] All elements in an indistinguishable set are not distinguishable from each other with every efficient algorithm.

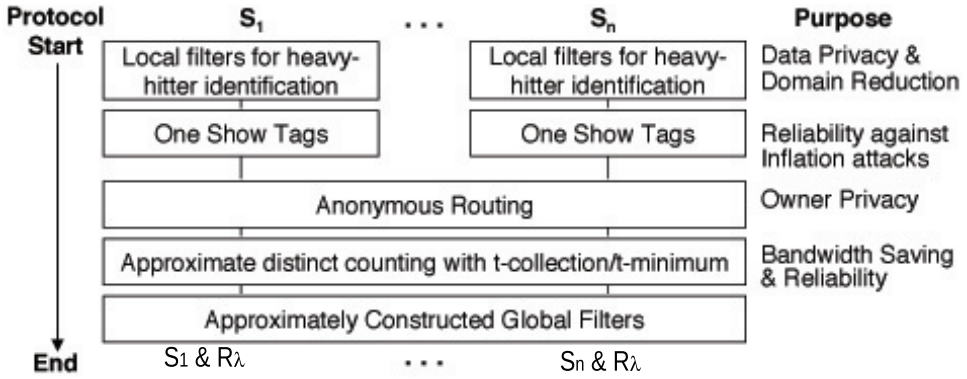


Figure 7.1: **Components of** HOTITEM-ID **protocol:** HOTITEM-ID defines how to efficiently compute an approximate representation of R_λ in distributed fashion. Each monitor i ($0 \leq i < M$) constructs local filters to approximately represent his private input set S_i, generates one-show tags for marked bits in filters, and sends a subset of those one-show tags to the network using anonymous routing. Those tags in the network are aggregated using a distributed counting protocol that estimates the number of distinct elements. At the end of the protocol, all monitors learn the global filters that approximate R_k. At the right side of the figure we list the purpose of each component.

7.2.1 Approximate Heavy-Hitter Detection

In our distributed signature detection system, we filter out the most content blocks by applying the local thresholds θ_G . However, the total number of content blocks from thousands of monitors will be still prohibitively large if thousands of monitors participate. In order to avoid exchanging the huge volume of data among distributed monitors, we utilize an *approximate heavy-hitter identification scheme* [45] instead of counting the number of monitors holding each possible content block and determining whether that content block is hot.

In this approximate heavy-hitter identification scheme, each monitor constructs a local filter. The monitors then combine their local filters to construct a global filter that represents R_λ; this global filter approximately identifies hot content blocks. We illustrate this process of local and global filter construction in Figure 7.2.

First, each player constructs a set of T *local filters*, which approximately represent his private input set. Let $h_1, \ldots, h_T : \{0,1\}^\omega \rightarrow \{1, \ldots, b\}$ be hash functions independent to each other. Each local filter contains b bits and $w_{i,q,j}$ represents the j-th bit in q-th local filter of monitor i. Monitor i ($0 \leq i < M$) computes each bit $w_{i,q,j} := 1 \Leftrightarrow \exists_{P \in S_i} h_q(P) = j$. The construction of local filters is similar to constructing a Bloom filter [8]. Indeed, we can use a combined Bloom filter instead of using separate T filters. We describe our approach using T filters in the interests of clarity.

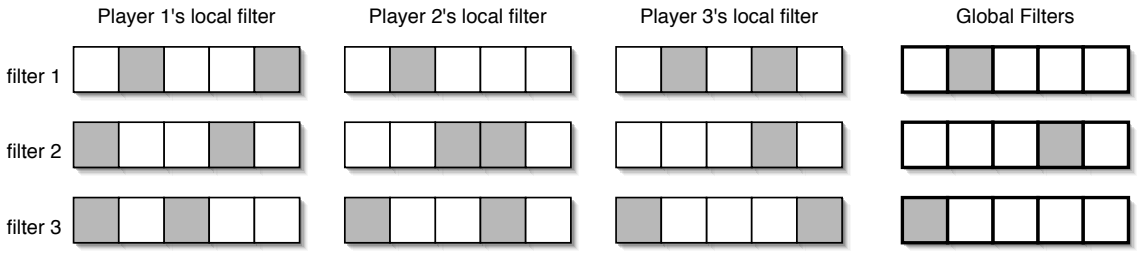


Figure 7.2: **Approximate Heavy Hitter Detection:** In our HOTITEM-ID protocol, each player i constructs a set of local filters from his private input set S_i (dark bits are 'hit'). The players then construct global filters using an approximate counting scheme; if a bit was hit by at least λ players, then it is 'hit' (dark) as well. If an element hashes to a dark bit in each of the global filters, then it is classified as hot.

The monitors then collaboratively combine their local filters into a set of *global filters*, using methods described below; global filters approximately represent the monitors' combined private input sets. If at least λ players set bits j of filter q to be 1, then the bit j in the global filter q, $x_{q,j} := 1$ ($0 \leq q < T$, $0 \leq j < b$). Otherwise, let $x_{q,j} := 0$. Given this global filter, P is hot with high probability if $x_{1,h_1(P)} = 1, \ldots, x_{T,h_T(P)} = 1$.

7.2.2 One-Show Tags

In order to construct the global filters, we must count how many monitors set each bit in their local filters. In order to prevent the inflation attack while preserving the owner privacy, the HOTITEM-ID protocol forces each monitor to 'vote' at most once with *anonymous one-show tags*.

If a monitor has set a bit, it constructs a tag for that bit; all monitors then count the number of valid tags for each bit to construct the global filters. We require that one-show tags posses the following properties: (a) no PPT adversary can construct more than one valid tag for any bit with non-negligible probability; (b) any monitor can detect tags that are invalid, or not associated with a particular bit, with overwhelming probability; (c) for every bit, the tags constructed by any two monitors are distinct, with overwhelming probability; (d) no PPT adversary can distinguish the monitor that constructed it, with probability non-negligibly different than $\frac{1}{(1-c)M}$, where M is the number of honest monitors and c is the fraction of malicious monitors.

The anonymous one-show tags are constructed by extending any group signature scheme. A group signature scheme allows each member in a group to sign messages on behalf of the

group. Given these signatures, no player or coalition of members (except the trusted *group manager*) can distinguish the member that produced any signature, nor can they determine if two signatures were produced by the same group member. In our work, we extend the group signature scheme of Boneh and Shacham [11] to construct the one-show tags. The one-show tag is lightweight, requiring 1539 bits.

In the Boneh and Shacham group signature scheme, the group public key is $\mathsf{pk} = \{g_1, g_2, w\}$, where $g_1 \in G_1$, $g_2 \in G_2$, and $w = g_2^{\mathsf{sk}}$ for $\mathsf{sk} \leftarrow Z_p^*$. ($p$ can be taken to be a 170-bit prime, and the elements of G_1, G_2 can be represented in 171 bits [11].) The trusted group manager holds the group secret key sk. Each user i ($1 \leq i \leq n$) has a private key $s_i = \{A_i, x_i\}$, where $x_i \in Z_p^*$, $A_i \in G_1$. Using his private key, each player may sign a message, using a variant of the Fiat-Shamir heuristic [47], by proving knowledge of a pair $\{A_i, x_i\}$ such that $A_i^{x_i+\mathsf{sk}} = g_1$.

We modify this group signature scheme to include provably correct one-show values, making each signature a one-show tag. Each monitor i ($1 \leq i \leq n$) constructs a one-show tag for bucket j of filter q ($1 \leq q \leq T, 1 \leq j \leq b$) by: (1) signing the message $q||j$ essentially as in the original signature scheme; (2) computing two additional values to enable the recipient monitor to compute the one-show value.

Each monitor generates $g_{q,j} \in G_2$ by an agreed-on scheme. We utilize the same bilinear mapping e as in the computation of the main signature, as well as the intermediate value v, α, r_α computed as intermediate values utilized in the main signature. Monitor i computes these additional elements for a one-show tag:

$$\begin{aligned} T_3 &= e(r_\alpha v, g_{q,j}) \\ T_4 &= e(\alpha v, g_{q,j}) \end{aligned}$$

The sole change to the original signature scheme is that the challenge value c is computed as $c = H(\mathsf{pk}, q||j, r, T_1, T_2, T_3, T_4, R_1, R_2, R_3)$. The recipient conducts all the validity checks specified in the Boneh/Shacham signature scheme, as well as the following additional check, derived from a proof of discrete logarithm equality [23]:

$$e(s_\alpha v, g_{q,j}) = T_4^c T_3$$

We define the one-show value as $e(A_i, g_{q,j})$; note that this value cannot be constructed by any player other than i and that player i can construct exactly one such value. To compute

this value, the signature recipient computes:

$$\left(e(-cT_2 s_\alpha v, g_{q,j})T_3\right)^{\frac{1}{c}}$$

The additional zero-knowledge proof required for the one-show tag construction is efficient, and thus our one-show tag construction and verification is nearly as efficient as the original group signature scheme. Note that these one-show tags are unlinkable, anonymous, and can be verified by all players who hold the group public key.

The parameter $g_{q,j}$, used to construct a one-show value associated with bucket j of filter T ($1 \leq q \leq T$, $1 \leq j \leq b$), can be efficiently constructed in a variety of ways such that no player knows its discrete logarithm. In this section, we briefly describe one such approach.

Let PRNG be a pseudo-random number generator with range G_2 [82]. Let the ℓth element output from this PRNG on seed σ be denoted σ_ℓ. If $\sigma^2_{(q-1)b+(j-1)}$ is a generator of G_2, then $g_{q,j} := \sigma^2_{(q-1)b+(j-1)}$. If this is not the case, then if $\left(h(\sigma_{(q-1)b+(j-1)})\right)^2$ is a generator of G_2, then $g_{q,j} := \left(h(\sigma_{(q-1)b+(j-1)})\right)^2$. If this is not the case, then repeat the process of hashing and testing until an element is found that is a generator of G_2. As a large portion of G_2 are generators, a low number of repetitions are generally needed.

Note that σ can be chosen anew each time the protocol is run to prevent reuse of one-show tags. For example, it might be generated from a common timestamp.

7.2.3 Approximate Distinct Element Counting

We can now securely identify hot items by utilizing global filters and anonymous one-show tags. However, exactly counting the number of valid, distinct one-show tags is inefficient. HOTITEM-ID performs the task of approximate counting of distinct tags through an efficient algorithm for *approximate distinct element counting* [6]. The HOTITEM-ID protocol provides even more efficiency by estimating directly, through a modified use of this approximation, whether there are more or fewer than λ tags for a bit; this is the task that must be performed to construct the global filters.

Let $\mathcal{H} : \{0,1\}^* \rightarrow \{0,1\}^\omega$ be a collision-resistant cryptographic hash function. Let $v_1, \ldots, v_t$ represent valid one-show tags that, when hashed with $\mathcal{H}$, have the smallest values of any tags in the set. The monitors can perform this tag collection task often without even examining every tag. For example, any tag with value greater than $\frac{t2^\omega}{\lambda}$ may be immediately discarded without affecting the approximation. We call this approximate tag counting technique *t-collection*.

There may be a large gap between the frequency of worm content blocks and legitimate traffic content blocks. Random scanning worms contact a large number of networks during a measurement interval. Even the localized scanning worms attempt to contact a large number of IP addresses in a local network. However, content blocks from legitimate traffic are shared among only a small number of IP addresses. Thus, to gain greater efficiency in detecting bits with at least λ one-show tags, we adjust the t-collection protocol such that each monitor attempts to collect t tags with hash values of at most $(1+\gamma)\frac{t2^{\omega}}{\lambda}$. The γ is a constant based on the size of the 'gap' between the frequency of worm content blocks and legitimate content blocks, and this gap needs to be determined based on the analysis of traffic. We gain efficiency by reducing t, while retaining accuracy.

7.2.4 Anonymous Communication

We now apply our tools to a network structure to complete the protocol. In order to disassociate the anonymous one-show tags from their origins in the network, each monitor anonymously routes its tags to ρ randomly chosen nodes. The constant ρ can be varied to achieve greater or lesser degrees of robustness; at least one participating monitor should receive each monitor's tags. We require an anonymous routing scheme that allows any monitor to send a message to a uniformly randomly selected monitor, without revealing the source to any intermediate routing node. Simple and lightweight schemes can be used, as HOTITEM-ID requires only that each monitor send anonymous messages to a uniformly selected destination node, without revealing who sent the message. Some previously proposed anonymous networking schemes include [24, 22, 111, 112, 84].

7.2.5 Distributed One-Show Tag Collection

Once all the monitors have anonymously received the initial set of one-show tags, they count the number of tags associated with each bit of the filters. The HOTITEM-ID protocol requires each monitor send messages to each of its neighbors, who form a connected graph. For clarity of presentation, our protocols assume synchronous communication, but this can be easily adapted to an asynchronous model.

Each monitor maintains a set of at most t valid tags which have hash values at most $(1+\gamma)\frac{t2^{\omega}}{\lambda}$. Upon learning a new tag that will be added to the set, a monitor sends the new tag to all of its neighbors. When a monitor has collected t tags with hash values of at most $(1+\gamma)\frac{t2^{\omega}}{\lambda}$, and sent each of these tags to its neighbors, it ends its participation

in the distributed t-collection protocol. If there do not exist t such small-hash-value tags, the protocol must continue until it converges. This t-collection process ends after at most $log_{\psi}(M)$ steps if ψ is the average number of neighbors. Note that these protocols can be executed in parallel for each bit of the global filters.

7.2.6 Putting HOTITEM-ID to Work

Each monitor has a private key allowing it to construct one-show tags. These keys are distributed by a trusted 'group manager'.

1. Each monitor constructs T local heavy-hitter identification filters.
2. Each monitor constructs a one-show tag for each bit in the filters set to 1.
3. If the hash ($\mathcal{H}$) of a tag is at most $(1+\gamma)\frac{t2^{\omega}}{\lambda}$, the monitor anonymously sends the tag to ρ randomly chosen nodes.
4. All monitors perform the distributed t-collection protocol.
5. If monitors collect t valid, distinct tags for a bit of a filter, they conclude that the value of corresponding bit in the global filter is 1.
6. Using the global filter, each monitor can determine if their own payload pattern strings are hot. The suspicious payload identification completes.
7. Each monitor generates signatures based on the result and publishes.

7.2.7 Correctness Analysis of HOTITEM-ID

In this section, we show that, given certain choices of the parameters b and T, HOTITEM-ID identifies hot items with high probability. In analyzing this protocol, we must consider both false positives and false negatives. Errors may be caused by inaccurate approximate distinct element counting, a badly constructed filter, or a combination of the two.

Error from Approximate Distinct-Element Counting. Bar-Yossef *et al.* [6] shows that if $h(v)$ is the tth smallest hash value in a set S, where $h : \{0,1\}^* \rightarrow \{0,1\}^{\kappa}$, then the estimate of the set size $|S|$ is $\frac{t2^{\kappa}}{h(v)}$. By computing the median of $O\left(\lg \frac{1}{\delta_1}\right)$ such estimates, with computationally independent hash functions, we can obtain $\overline{|S|}$, an (ϵ, δ_1)-approximation of the set size.

Theorem 1. *For any $\epsilon, \delta_1 > 0$, the approximate distinct-element counting algorithm (ϵ, δ_1)-approximates $|S|$. That is,*

$$Pr\left[\overline{|S|} > (1+\epsilon)|S| \vee \overline{|S|} < (1-\epsilon)|S|\right] \leq \delta_1$$

Proof. Proof is given by Bar-Yossef *et al.* in [6]. □

Our HOTITEM-ID employes the approximate distinct-element counting algorithm noticing that, if there exist at least t elements with hash values at most $\frac{t2^\kappa}{\lambda}$, the estimate of $|S|$ is at least λ.

Corollary 2. *If there exist t values $v_1, \ldots, v_t \in S$ such that $\forall_{\ell \in [t]}\, h(v_\ell) \leq \frac{t2^\kappa}{\lambda}$, then the approximate distinct element counting algorithm of [6] will estimate that $|S| \geq \lambda$.*

Proof. Let $w = \max_{\ell \in [t]}\{h(v_\ell)\}$. The approximation algorithm specifies that $\overline{|S|} = \frac{t2^\kappa}{w}$. As $w \leq \frac{t2^\kappa}{\lambda}$, then $\overline{|S|} \geq \lambda$. □

Error from Filters. Now we consider the probability that an item a, which appears in less than λ' monitors' private input sets (*e.g.* $F_a < \lambda'$), is identified as a λ'-threshold hot item. We set $\lambda' := \frac{\lambda}{1+\epsilon}$, so as to account for the allowed inaccuracy in the approximate counting algorithm.

Let's say j'-th hash value of a cold item a is j ($0 \leq j < b$). If the corresponding bucket j in the j'-th filter was hit by $\lambda' - F_a$ monitors who do not hold the item a due to hash collisions, the cold item a will be identified by the filter. Moreover, as malicious participants may claim to hit all buckets, a minimum of $\lambda' - F_a - cM$ honest monitors must hit the bucket j to cause an error regarding a.

We assume that each honest monitor has a maximum of m items in their private input set. Using every element of each players' private input set, each group of $\lambda' - F_a - cM$ honest monitors may hit at most m buckets a suficcient number of times to introduce an error caused by collisions. There are $M - F_a - cM$ honest monitors who do not hold a, and thus $\lfloor \frac{M-F_a-cM}{\lambda'-F_a-cM} \rfloor$ groups of monitors that can hit m buckets per group enough times to allow danger of an error. Note that any group of fewer than $\lambda' - F_a - cM$ monitors cannot hit any particular bucket a sufficient number of times to cause a total of $\lambda' - F_a$ hits; each monitor may only hit any particular bucket at most once.

Thus, at most $m\lfloor \frac{M-F_a-cM}{\lambda'-F_a-cM} \rfloor$ buckets of each filter can be 'unsafe'; if a is not mapped to one of those buckets, then there is no possibility of error from the malfunctioning of the filter.

As $h_{j'}(a)$ is distributed computationally indistinguishably from uniformly over $[b]$, we may thus bound the probability that bucket $h_{j'}(a)$ is erroneously designated as 'hot':

$$\Pr\left[a \text{ is identified as } \lambda'\text{-hot by one filter}\right] \quad \leq \quad \frac{m\left\lfloor \frac{M-F_a-cM}{\lambda'-F_a-cM} \right\rfloor}{b}$$

Combined Error. We conside the two sources of error together. There are two possible error types: *false positives* in which cold items are identified as hot, and *false negatives* in which hot items are not identified.

Theorem 3. *Given the false positive rate δ_+ and the false negative rate δ_-, error bounds ϵ and β, the upper limit of the number of malicious participants cM. Let b, t, T, ρ be chosen as the following: $t := \lceil \frac{96}{\epsilon^2} \rceil$, $\rho := O\left(\lg \frac{2}{\delta_-}\right)$, $\alpha := O\left(\lg \frac{2}{\delta_-}\right)$, b and T are chosen to minimize $b \times T$, and at the same time, satisfy $\left(\frac{m\left\lfloor \frac{M-\beta\lambda-cM}{\frac{\lambda}{1+\epsilon}-\beta\lambda-cM} \right\rfloor}{b} + \frac{\delta_-}{T} \right)^T < \delta_+$.*

In the HOTITEM-ID *protocol, with probability at least $1-\delta_+$, every element a that appears in $F_a < \beta k$ players' private input sets is not identified as a λ-threshold hot item.*

In the HOTITEM-ID *protocol, with probability at least $1-\delta_-$, every element a that appears in $F_a \geq \frac{\lambda}{1-\epsilon}$ players' private input sets is identified as a λ-threshold hot item.*

Proof. The probability that an element a, which appears in $F_a < \frac{\lambda}{1-\epsilon}$ players' private input sets, is not identified as a λ-threshold hot item can be bounded as follows. Note that we have set $\lambda' = \frac{\lambda}{1+\epsilon}$, to account for the allowed tolerance in approximate counting:

$$
\begin{aligned}
\Pr\left[a \text{ is identified as } \lambda\text{-hot}\right] &\leq \Pr[\text{in all filters, element } a \text{ is identified as } \lambda'\text{-hot} \vee \\
&\quad \text{set of size} < \lambda' \text{ approximated as } \geq \lambda] \\
&= \prod_{j'=1}^{T} \Pr[\text{in filter } j', \text{ element } a \text{ is identified as } \lambda'\text{-hot} \vee \\
&\quad \text{set of size} < \lambda' = \frac{k}{1+\epsilon} \text{ approximated as } \geq \lambda] \\
&\leq \prod_{j'=1}^{T} \left(\frac{m \left\lfloor \frac{n - F_a - cM}{\lambda' - F_a - cM} \right\rfloor}{b} + \delta_1 \right) \\
&= \left(\frac{m \left\lfloor \frac{n - F_a - cM}{\lambda' - F_a - cM} \right\rfloor}{b} + \delta_1 \right)^T \\
&= \left(\frac{m \left\lfloor \frac{n - F_a - cM}{\frac{\lambda}{1+\epsilon} - F_a - cM} \right\rfloor}{b} + \delta_1 \right)^T
\end{aligned}
$$

If an element a, which appears in $F_a \geq \frac{\lambda}{1-\epsilon}$ players' private input sets, is not identified as a λ-threshold hot item, it is due to error in the set-counting approximation. When the number of hits for every bucket in a filter is counted exactly, there can be no false negatives. Thus, we may bound the probability of a false negative as follows:

$$
\begin{aligned}
&\Pr\left[a \text{ is not identified as } \lambda\text{-hot}\right] \\
= \;& \Pr\left[\text{in at least one filter, a set of size } \geq \tfrac{\lambda}{1-\epsilon} \text{ approximated as } < \lambda\right] \\
\leq \;& \textstyle\sum_{j'=1}^{T} \Pr\left[\text{in filter } j', \text{ a set of size } \geq \tfrac{\lambda}{1-\epsilon} \text{ approximated as } < \lambda\right] \\
\leq \;& \textstyle\sum_{j'=1}^{T} \delta_1 \\
= \;& \delta_1 T
\end{aligned}
$$

□

Choice of Constants. Given our analysis, we may outline how to choose the constants δ_1, t, b, T based on the parameters $\epsilon, \delta_-, c, M, m, \delta_d, \beta$.

- δ_1**.** Recall that the probability of a false negative, for an element a which appears in at least $F_a \geq \frac{\lambda}{1-\epsilon}$ players' private input sets, is required to be at most δ_-. We may then

choose δ_1 as follows:

$$\begin{aligned} \Pr\left[a \text{ is not identified as } \lambda\text{-hot}\right] &= \delta_- \\ &\leq \delta_1 T \\ \delta_1 &:= \frac{\delta_-}{T} \end{aligned}$$

- b, T**.** Given this assignment of δ_1, we may simplify our bound on the probability of a false positive on an element a, which appears in $\beta\lambda < \frac{\lambda}{1+\epsilon}$ players' private input sets, as follows:

$$\begin{aligned} \Pr\left[a \text{ is identified as } \lambda\text{-hot}\right] &\leq \left(\frac{m \left\lfloor \frac{M - F_a - cM}{\frac{\lambda}{1+\epsilon} - f_a - cM} \right\rfloor}{b} + \delta_1 \right)^T \\ &\leq \left(\frac{m \left\lfloor \frac{M - \beta\lambda - cM}{\frac{\lambda}{1+\epsilon} - \beta\lambda - cM} \right\rfloor}{b} + \frac{\delta_-}{T} \right)^T \end{aligned}$$

  We choose b, T so as to minimize $b \times T$, while satisfying the above constraint.

- t, α**.** In the approximate distinct element counting algorithm in [6], $t := \lceil \frac{96}{\epsilon^2} \rceil$, $\alpha := O\left(\lg \frac{2}{\delta_-}\right)$. In practice, one may safely choose to retain $t < \lceil \frac{96}{\epsilon^2} \rceil$ smallest values per parallel execution, and run only one parallel execution, while retaining a confidence bound of δ_1. We found that $t := 25$ was sufficient.

  Note that, when running very small examples, or with very high accuracy requirements, one may obtain an assignment for t that is $\geq \lambda$. In this case, simply set $t := \lambda$, and note that $\epsilon, \delta_1 = 0$; each player is now collecting a sufficient number of signatures so as to determine, without error, whether any particular bucket was hit by at least λ distinct players.

7.2.8 Privacy in HOTITEM-ID

Theorem 4. *Assume that one-show tags are unlinkable and that the anonymous communication system is secure such that no coalition of adversaries can distinguish which honest player sent any given anonymous message with probability more than negligibly different from a random guess. In the* HOTITEM-ID *protocol, for any payload P, no coalition of at most cM malicious players can gain more than a negligible advantage in determining if $P \in S_i$, for any given honest player i ($0 \leq i < M$).*

When considering data privacy, we wish to prove that the non-hot items in $S_1 \cup \cdots \cup S_n$ remain hidden from adversaries. For example, in the HOTITEM-ID protocol, no player or coalition of at most cM malicious players may gain more than an expected $I = \frac{tmM}{k} \lg b$ bits of information about all players' inputs $\bigcup_{i=1}^{n} S_i$, when m is the maximum possible size of S_i for $1 \leq i \leq M$. We prove a tighter bound on the degree to which each payload is hidden in a crowd of *indistinguishable elements*. Two elements are indistinguishable if an attacker cannot distinguish which one was in players' private input sets based on the information gained in HOTITEM-ID. For a payload pattern P, its indistinguishable set consists of all payload patterns indistinguishable from it. To provide data privacy, we wish for non-hot items to have large indistinguishable sets.

Theorem 5. *In the* HOTITEM-ID *protocol, each payload P, which appears in F_P distinct players' private input sets, has an indistinguishable set of expected size*

$\Phi(F_P) = \sum_{\ell=1}^{T} \binom{T}{\ell} \left(1 - \frac{t}{k}\right)^{F_a(T-\ell)} \left(1 - \left(1 - \frac{t}{k}\right)^{F_a}\right)^{\ell} \frac{|M|}{b^{\ell}}$.

Proof of these theorems is given in [69]. We graph $\frac{\Phi(F_P)}{|M|}$ in Figure 7.3 when $T = 5$ and $b = 2000$. Note that if P appears in only a few players' private input sets, a very large proportion of the domain is indistinguishable from P. As F_P approaches $\frac{\lambda}{t}$, the size of the indistinguishable set decreases; this character ensures that truly rare items are highly protected. As t is a constant, independent of $|M|$, while λ will often grow with the size of the network, we see that protection for cold items generally increases as the network increases in size.

Note that in our correctness analysis in Section 7.2.7, we found that $t \geq 25$ was sufficient in practice. In order to prevent cold items from being disclosed by errors, we have to choose λ to be large enough so that $\frac{\lambda}{t} >> F_P$ for most of the cold items P. In other words, our HOTITEM-ID is a viable solution for privacy protection only when the number of participants is as large as thousands and therefore we can choose a large λ such as hundreds.

7.3 Distributed Worm Signature Detection

We now apply the HOTITEM-ID protocol to detect the signature of a worm by correlating the content blocks discovered by individual Autograph monitors. There are two possibilities in placing an Autograph monitor. First, we can install an Autograph monitor in an edge network's DMZ as described in Chapter 3. Second, we implement Autograph's suspicious flow selection and content-based signature generation stages in an end host. The *end-host*

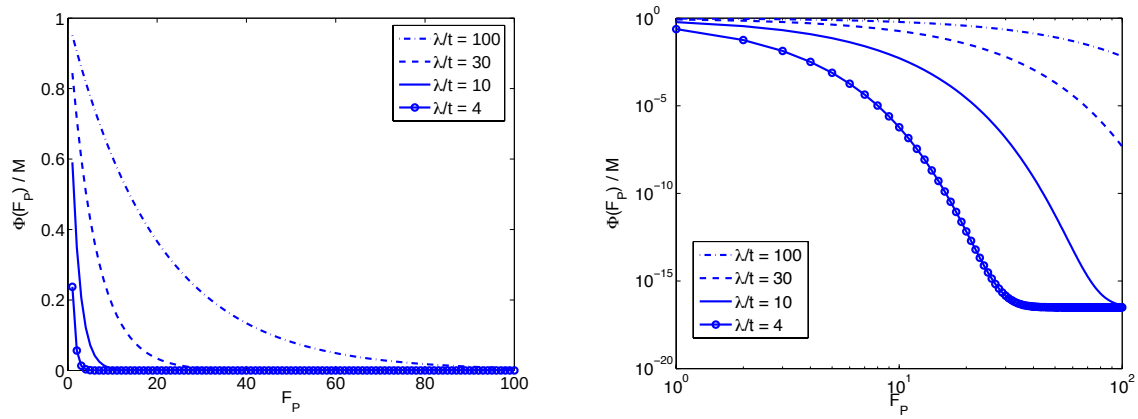


Figure 7.3: The degree of data privacy for an element with frequency F_P is $\frac{\Phi(F_P)}{|M|}$. We graph this function in **(a)**, showing the increase in protection for items that show up in few players' inputs. The same function is graphed in **(b)** on a logarithmic scale, for increased detail. Note that rare items are indistinguishable from a large proportion of the domain, giving them a large degree of data-privacy.

Autograph is installed in a dedicated network processor [21] or in the kernel of an end host to minimize the risk of fraudulent reports from already-compromised applications. End-host Autograph, thus, can perform flow reassembly similarly to the applications at the end host. Still, Autograph's port scanner detection must be done at the boundary of the network and the gateway of each subnet because port scanner detection needs to monitor each unsuccessful connection attempt to a non-existent host or service. This cannot be done at the end hosts. The list of port scanners is distributed to all end-host Autograph systems using the tattler mechanism.

The advantage of placing Autograph in an edge network's DMZ is that the monitor can possibly accumulate many suspicious flows and perform local content-prevalence analysis based on locally found suspicious flows. If a single monitor accumulates enough suspicious flows, the monitor generates a signature immediately. From a network administrator's perspective, this is easier to install and maintain than end-host Autograph.

An advantage of end-host Autograph is that we can perform the flow reassembly at the end host, and thus avoid attacks that exploit ambiguities in traffic stream as seen by the NIDS [106]. Exploitable ambiguities can arise in two different ways in Autograph's traffic analysis:

1. Without detailed knowledge of the end-host's protocol implementation, Autograph in the DMZ may be unable to determine how the end-host will treat a given sequence of packets if different implementations interpret the same stream of packets in different

ways. For example, when an end-host receives overlapping IP fragments that differ in the purported data for the overlapping region, some end-hosts may favor the data first received, others the portion of the overlapping fragment present in the lower fragment, and others the portion in the upper fragment.

2. Without detailed knowledge of the network topology between the DMZ and the end-host, Autograph in the DMZ may be unable to determine whether a given packet will even be seen by the end-host. For example, a packet seen by Autograph in the DMZ area that has a low Time-To-Live (TTL) field may or may not have sufficient hop count remaining to make it all the way to the end-host. If the Autograph uses such packets in its analysis, the analysis result may be incorrect.

Another advantage is that we can monitor worm activities inside a network. Many modern worms employ localized scanning to infect hosts in the same subnet quickly, but a monitoring system at the border of the network cannot detect such internal worm infection activities. With scanner detection at each subnet boundary, end-host Autograph can collect worm flows even when the worm launches attacks from an internal host. Weaver *et al.* [141] and Staniford [128] also discuss the importance of worm activity monitoring inside enterprise networks.

The disadvantage of end-host Autograph is obvious. The probability of receiving multiple attack flows is so low that content analysis and signature generation is difficult. Thus, collaboration among multiple end-host Autograph monitors is necessary in the signature generation process.

In this work, we focus on the end-host Autograph case. Thousands of end-host Autograph monitors are connected with a P2P multicast network. They receive port-scanner information from monitors installed in their network's DMZ, boundaries of each subnets, and network telescopes [86]. If a monitor receives a connection attempt from one of the blacklisted port-scanners, the monitor responds with the connection request, collects suspicious packets, and performs TCP flow reassembly. The resulting reassembled payloads are partitioned into content blocks using Autograph's COPP algorithm. The monitor's private dataset consists of those content blocks collected during the last τ minutes. Finally, all monitors collaboratively perform signature generation in our LSD and GSD frameworks with their private datasets.

7.3.1 Candidate Signature Advertisement (LSD)

If a content block appears more than θ_L times and more than one source IP addresses has sent the content block, the monitor advertises the content block as a candidate signature. Using a secret key sk_i and a group public key distributed by a trusted group manager, each monitor M_i constructs a one-show tag. The detail of one-show tag construction is described in [69]. The one-show tag is unlinkable, anonymous, and can be verified by all other monitors who hold the group public key. The same one-show tag cannot be constructed by any monitor other than M_i who holds sk_i.

The candidate signature with a corresponding one-show tag is sent to ρ randomly chosen monitors using an anonymous scheme such as AP3 [84] and Crowds [112]. The message delivery to a randomly chosen monitor is done as the followings:

1. When a monitor M_i wants to send a message to a randomly selected monitor M_j, M_i chooses another node (say M_k) randomly, and sends the message to M_k.

2. Upon receiving the message, M_k performs a weighted coin toss to decide whether to send the message to M_j, or to forward the message to another randomly chosen node. The decision to forward to another random node is made with probability p_f ($0.5 \leq p_f < 1$), the forwarding probability.

3. After $\frac{1}{(1-p_f)}$ hops on average, the message is delivered to M_j.

The mechanism obscures the originator M_i's identity from both the intended recipient M_j and any malicious nodes hoping to expose the originator's identity. Thus, the owner privacy is protected. Unlike AP3, we do not need an anonymous bi-directional communication channel. Therefore, the implementation is simpler and more light-weight than the original AP3 scheme.

Once one of the ρ monitors receives the message with a candidate signature, the message is multicasted to all other monitors. Upon receiving a multicast message, each monitor verifies the validity of the message using the group public key. Each monitor maintains a counter for each valid, advertised candidate signature. If a monitor receives more than λ_L advertisements on the same candidate signature, it accepts the candidate signature as a valid global signature for a currently active worm.

7.3.2 Suspicious Payload Sharing (GSD)

Once every d minutes, all monitors run the HOTITEM-ID protocol to identify content blocks that do not appear many times in a single dataset but do appear in many datasets. If a content blocks appear in more than λ_G monitors' private datasets, the content block is used to generate a new signature. The detailed procedure is described in Section 7.2.6. All monitors use the same set of hash functions, and t-collection parameters. Once a monitor finds one of its content blocks appears in more than λ_G monitors' data set, it generates a signature based on the content block, and publishes it using the same anonymous communication scheme used in the LSD implementation. Since all monitors already have the global filter, they can check the validity of the signature with the global filter and do not have to wait until multiple signature advertisement messages arrive.

For both GSD and LSD, we use a basic form of multicast communication in our implementation. Each monitor has ψ neighbors and they form a connected graph. When a monitor receives a message that is not received before, it forwards the message to all $\psi - 1$ neighbor monitors other than the message sender. A message can be delivered multiple times to a node along the different paths, but there is no loop because only newly found messages are forwarded to the neighbors. By employing a more efficient multicast protocol [60, 19] that delivers data along a multicast tree, we could avoid the duplicate message delivery. We leave this improvement as future work.

The signature detection speed of GSD and LSD was examined in Section 6.2. Since the one-show tag construction and verification can be done quickly, the lag for candidate signature distribution in LSD and payload sharing in GSD is governed by communication delay. The average path length to multicast a message is $O(\frac{1}{1-p_f} + log_\psi(M))$. In order to protect the owner privacy, the larger forwarding probability p_f is preferred. However, the larger p_f we use, the longer it takes to anonymously route each message.

Even though there could be thousands of content blocks in each monitor's private dataset, only a small fraction of one-show tags need to be sent to the network. In the next section, we measure the required bandwidth and the number of messages using real network traces.

7.4 Experimental Results

We now proceed to evaluate the efficiency and accuracy of GSD implemented with HOTITEM-ID with simulations. We are interested in the number of exchanged messages and the required bandwidth. Each message containing an one-show tag has to be verified by each

monitor. Even though the time to verify one message is small in our implementation, the total processing time would be prohibitively large if many messages need to be verified. The message size in HOTITEM-ID is large; each one-show tag requires 1,368 bits (171 Bytes). Thus, the bandwidth consumed by HOTITEM-ID is another concern in building a scalable system. Our experimental results support the efficiency and accuracy of HOTITEM-ID .

7.4.1 Simulation Method

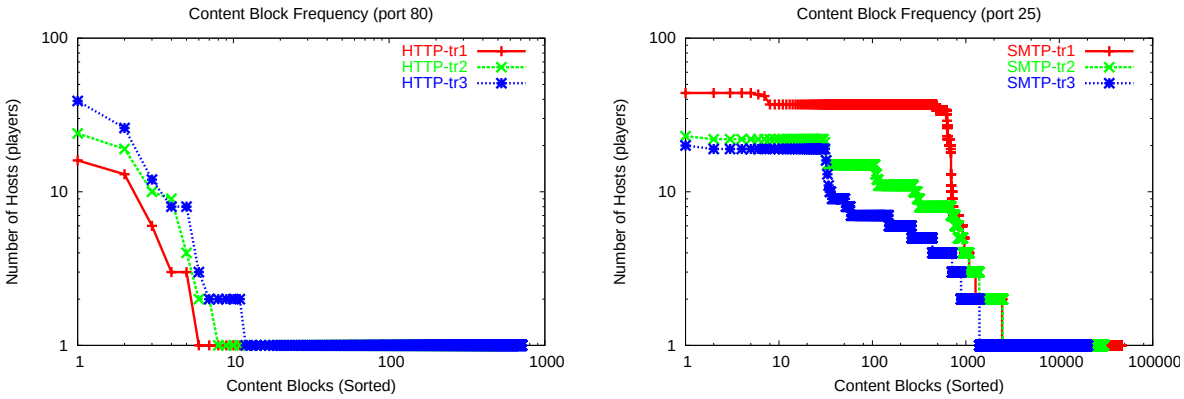
We measure the required bandwidth, the number of messages, and the false positive/negative rates by simulating signature generation during simulated worm propagation. We compare our protocol to a non-private naïve protocol, in which all content blocks are forwarded to all participating 1024 monitors. To save bandwidth and give a trivial measure of privacy against casual attacks [78], the naïve protocol hashes each content block with SHA-1 and exchanges only the hash values. When a monitor receives more than λ_G messages for a hash value, the monitor knows that the corresponding content block is λ_G -hot. The naïve protocol uses the same network topology and the same communication model as our HOTITEM-ID protocol.

In our experiments, we assume that 1024 end-host Autograph monitors inside a B-class network collaboratively generate a signature by performing GSD with HOTITEM-ID . The monitors receive port-scanner lists from monitors installed at the border of the network and each subnet. The monitor at the gateway monitors ICMP host/port unreachable messages and unanswered inbound SYN packets. If an external host has made unsuccessful connection attempts to more than 2 internal IP addresses, the monitor advertises the IP address to all 1024 end-host Autograph monitors. Once a flow is identified as a suspicious flow by an end-host Autograph monitor, the flow is held in the monitor's suspicious flow pool for $\tau = 1$ hour. Each end-host Autograph monitor constructs its private dataset by performing COPP. We use 64 bytes as the average content block size parameter.

Traces. In order to represent content blocks from misclassified, innocuous flows, we use network traces captured at the gateway of a campus network in Feb 18, 2005. The network has a class-B IP address space. The gateway has three 1 Gbps interfaces and incoming traffic is evenly distributed to the interface using per-flow load balancing. Our traces are collected by tapping one of the links. We split the traces based on the destination IP addresses and feed end-host Autograph with the splitted traces and collect content blocks generated for an hour. Most content blocks are from port 80 (`HTTP`) and port 25 (`SMTP`) traffic. Here, we present our experimental results with traffic of only those ports. Table 7.1 summarizes the traces used in our experiments. Misclassified SMTP flows tend to generate many content

name	duration	protocol	misclassified flows	unique con -tent blocks	content blocks appeared in more than 1 monitor
HTTP-tr1	1 hour	http	288	521	5
HTTP-tr2			312	652	6
HTTP-tr3			355	724	12
SMTP-tr1	1 hour	smtp	1489	46120	1489
SMTP-tr2			1322	38210	1521
SMTP-tr3			1202	25284	1078

Table 7.1: Traces and the number of generated content blocks



(1) HTTP (port 80) **(2) SMTP (port 25)**

Figure 7.4: **Number of hosts vs. Innocuous content blocks:** Number of unique hosts that report each content block. The content blocks are generated from suspicious flows collected for 1 hour time window.

blocks because they often contain MIME-encoded long messages. Even though the total number of unique content blocks is large, only a small fraction of them appear in multiple monitors. Only 1.6% (HTTP-tr3) and 3.2% (SMTP-tr1) of content blocks appear at more than one host.

Figure 7.4 (a) and (b) show the number of hosts who generate each content block for the misclassified, innocuous HTTP and SMTP traffic, respectively. Through manual examination, we determined that content blocks that appeared at more than one host, belong to web crawlers' traffic or http header information (HTTP), or spamming activities (SMTP). We consider all these content blocks to be innocuous. None of the innocuous content blocks appear at more than 45 hosts.

Simulated Worm Flows. Once an internal host is infected, localized scanning worms can easily penetrate the entire network. The fast scanning pattern is a prominent feature across many worms. Assuming a localized scanning worm that targets hosts in the B-class

IP address space, we inject simulated worm traffic into the suspicious flow pools of 200 end-host Autograph monitors. Those 200 monitors receive exactly one worm flow each, and generate 10 content blocks from the worm flow. All 10 content blocks are invariant across worm flows, and ideally, our system must find all those content blocks as hot content blocks.

Metrics. We evaluate the performance of our HOTITEM-ID -based GSD in terms of the number of received messages per monitor, and required bandwidth per monitor, while varying the t-collection parameters, $t \in [3, 5, 10]$ (the number of tags to collect) and $\gamma \in [0, 0.2, 0.5]$ (the gap factor), and the average number of neighbors $\psi \in [5, 10]$. We count the number of received messages at each monitor and report the average. Since all messages are delivered to every monitor, the variance in the number of messages is small. We measure the false positives as the number of innocuous content blocks identified as to be hot. The false negative rate is the fraction of worm content blocks that were not identified as to be hot.

The number of filters T and the size of each filter b must be chosen based on the estimated total number of content blocks in the system, which is measured by analyzing normal traffic. Too small b and T cause many hash collisions in approximate heavy-hitter detection, and thus incur many false positives. In our experiment, we utilized the parameters $b := 606$, $T := 5$ for HTTP traces, and $b := 4545$, $T := 5$ for SMTP traces according to the guidelines in [69]. With the parameters, the false positive rate δ_+ is lower than 0.01 if a content block appears in less than 50 monitors' private datasets in the presence of $1\% \cdot 1024$ malicious monitors. The false negative rate δ_- is lower than 0.01 if a content block appears in more than λ_G monitors' private datasets.

7.4.2 Bandwidth Consumption and Accuracy

We ran the HOTITEM-ID protocol to find content blocks that appear in at least 100 monitors' private dataset (λ_G = 100).

False positives. In our experiments, there was no false positives except for `SMTP-tr1` with the t-collection parameters, $t \leq 3$ (the number of tags to be collected) and $\gamma \geq 0.5$ (the gap factor). We observed two innocuous content blocks mistakenly categorized as λ_G-hot in the experiment using `SMTP-tr1` with $t \leq 3$ and $\gamma \geq 0.5$. The false positives are due to content blocks generated from a spam e-mail. The e-mail was sent to 44 different hosts. The γ parameter must be chosen based on the difference between the frequency of innocuous content blocks and the expected frequency of worm content blocks. In the `SMTP-tr1` trace, hundreds of innocuous content blocks were observed by more than 30 monitors, and ten

		HTTP-tr3						SMTP-tr1					
		# of messages			bandwidth			# of messages			bandwidth		
		t=3	t=5	t=10	t=3	t=5	t=10	t=3	t=5	t=10	t=3	t=5	t=10
ψ=5	naïve	1 (11,399msgs)			1 (228KB)			1 (300,757msgs)			1 (6015KB)		
	γ=0	(0.044)	0.075	0.151	(0.370)	0.635	1.274	(0.026)	(0.039)	0.073	(0.222)	(0.326)	0.616
	γ=0.2	**0.048**	0.081	0.162	**0.408**	0.685	1.366	(0.032)	(0.048)	0.088	(0.274)	(0.402)	0.740
	γ=0.5	0.052	0.088	0.177	0.442	0.745	1.492	0.042	**0.062**	0.117	0.354	**0.522**	0.986
ψ=10	naïve	1 (25,643msgs)			1 (513KB)			1 (676,612msgs)			1 (13,532KB)		
	γ=0	(0.038)	0.068	0.141	(0.325)	0.572	1.193	(0.017)	(0.021)	0.065	(0.145)	(0.178)	0.548
	γ=0.2	**0.042**	0.071	0.146	**0.353**	0.596	1.230	(0.022)	(0.028)	0.068	(0.187)	(0.234)	0.572
	γ=0.5	0.044	0.074	0.149	0.371	0.623	1.256	0.036	**0.062**	0.124	0.305	**0.523**	1.048

Figure 7.5: **Normalized bandwidth consumption per player in performing hot item identification ($\lambda_G = 100$):** Numbers in parentheses denote cases with false negatives. Underlines denote cases with false positives. Numbers in bold face fonts are the most efficient but accurate cases in each set of experiments.

payloads among them appeared in more than 40 different monitors, which is close to the hot item identification threshold $\lambda_G = 100$ used in our experiments. A larger γ causes those innocuous content blocks to be sampled more often and results in false positives. In the `HTTP-tr3` experiments, we did not experience any false positive case even with $\gamma \geq 0.5$, because there are only a couple of content blocks that appeared in more than 30 monitors' private dataset.

False negatives. Unfortunately, we observed two false negative cases when $\gamma = 0$ and $t \leq 3$ in the experiments with `http` traces, and when $\gamma \leq 0.2$ and $t \leq 5$ in the experiments with `smtp` traces. The false positive rate is higher than what we target. This is because the t value is too small to tolerate the error from approximate distinct element counting. A higher t value reduces the error rate.

Efficiency. Our HOTITEM-ID protocol scales better than the naïve protocol. We present our comparison of the required bandwidth and messages in Figure 7.5. For easy comparison, we normalize the measured values against the results from the naïve protocol. Our HOTITEM-ID implementation, based on an efficient group signature scheme [11], requires 173 bytes per message while the naïve protocol utilizes only 20 bytes per message for a SHA-1 hash value. However, our protocol requires only a small number of message transmissions; as a result, HOTITEM-ID used only 35% and 52% of the bandwidth used by the naïve protocol in the `HTTP-tr3` and `SMTP-tr1` experiments, respectively when the appropriate γ and t parameters are used.

In `HTTP-tr3`, there are only 724 unique content blocks globally while `SMTP-tr1` generates 46,120 unique content blocks. As a result, monitors need to exchange more messages in the experiments with `SMTP-tr1` than with `HTTP-tr3`. Note that there is duplicated mes-

sage delivery because we use a basic form of multicast that floods each message to all ψ neighbors. Thus, the number of messages each node receives is larger than the actual number of unique messages. A larger ψ supports fast message broadcast to all monitors, but results in more duplicated messages in the basic multicast.

As expected, the number of messages increases as γ and t increase. However, there is a trade-off between efficiency and correctness. Numbers in parentheses in Figure 7.5 denote cases with false negatives. Numbers with underlines denote cases with false positives. Smaller γ and t values reduce the communication cost but may result in false negatives. Our experiments show that to ensure correctness while retaining efficiency, we may set $\gamma = 0.2, t = 3$ (HTTP) and $\gamma = 0.5, t = 5$ (SMTP) for the traffic pattern in our traces. The numbers in the bold face font in Figure 7.5 represent the most efficient cases. Without analyzing more traffic traces from other time periods and other sites, we are unable to claim that the set of parameters is valid for traffic patterns of other protocols.

Our HOTITEM-ID protocol scales better than the naïve protocol; as problems increase in size, our protocol becomes more attractive. For example, when 10240 monitors participate in worm signature generation and we need only 10% of them to catch worm traffic ($\lambda = 1000$), our HOTITEM-ID protocol uses less than 6% of the bandwidth used by naïve protocol.

7.5 Summary

In this chapter, we use an efficient, secure, and robust privacy-preserving distributed hot item identification protocol (HOTITEM-ID) for suspicious payload identification across thousands of monitors. The HOTITEM-ID protocol protects owner- and data-privacy by utilizing an approximate heavy-hitter identification scheme, one-show tags, an approximate distinct item counting scheme, and anonymous routing techniques. The protocol provides probabilistic privacy to support computation across thousands of distributed nodes. However, the probabilistic privacy is sufficient for our suspicious payload pattern identification problem. We implemented the protocol and simulated distributed suspicious payload pattern identification process by injecting worm traffic to a real network trace. Our experimental results support that we can efficiently perform the distributed signature generation with the HOTITEM-ID protocol while preserving the privacy in the presence of thousands monitors participating.

Chapter 8

Preserving Privacy using Privacy-Preserving Multiset Operations

The privacy-preserving payload sharing technique presented in the previous chapter, enables an efficient, secure, and robust privacy-preserving payload sharing. However, the technique uses probablistic approaches so the correctness and the privacy protection depends on a large number of participating monitors. Thus we expect that the technique proposed in the previous chapter is viable only when thousands of monitors participate in payload sharing (See Section 7.2.8). If the number of monitors is small, we cannot choose the threshold large enough to protect the data privacy of 'cold' content blocks while distinguish 'hot' and 'cold' content blocks accurately. The system is not able to provide sufficient accuracy or privacy protection for our new scenario.

This chapter introduces a privacy-preserving distributed signature generation system that works well with about a hundred of monitors. The system is based on the privacy-preserving multiset operation framework of Kissner and Song [70]. A semantically secure, threshold homomorphic cryptosystem allows collaboration without a trusted third-party, while preventing disclosure of rare, non-worm payload patterns. Unfortunately, the strong privacy guarantee comes at the cost of expensive cryptographic operations and multiple message exchanges among participating monitors. We analyze the complexity of the basic protocol that directly applies the privacy-preserving multiset operation framework, and then propose extended privacy-preserving signature generation protocols that use approximate counting techniques to improve performance. The results of the experiment with a prototype implementation of the extended protocols show that the system that connects 128 monitors can determine mutually observed payload patterns within 5 minutes even with

the currently available computer hardware. Those payload patterns are used for payload pattern based signature generation.

We formalize our signature detection problem in the next section. In Section 8.2, we summarize the threshold homomorphic cryptosystem and the privacy-preserving multiset operation framework that are basis of our implementation. In Section 8.3, we describe our algorithms used in our system, including the THRESHOLD-SET-UNION-HBC protocol developed by Kissner and Song [70]. We present the implementation of our system in Section 8.4, and examine its performance and scalability in Section 8.5.

8.1 Problem Definition

In this chapter we design a distributed payload sharing system of M monitors following the GSD framework defined in Chapter 6. Each monitor shares a content block P with other monitors only when its prevalence measure $f(P)$ exceeds θ_G. Only the content blocks that are shared by more than λ_G monitors are used as signatures.

In the system, there are at most c honest-but-curious monitors. Honest-but-curious monitors act according to their prescribed action in the protocols. However, they may attempt to gain more information other than what can be deduced from the result of the protocol.

The system protects the privacy of exchanged data and its owner monitor against those c honest-but-curious monitors. No monitor or coalition of c honest-but-curious monitors gains more information about other monitors' private datasets, other than 1) the set of λ_G-hot content blocks, and 2) the number of content blocks reported by each monitor.

The original protocol proposed by Kissner and Song [70] prevents honest-but-curious adversaries even from inferring the number of items (content blocks) in the data set of each player (monitor). The protocol protects the information by enforcing all players to report the same number of items. To avoid the case that a player has more than the predefined number of items, the protocol assumes a large value for the private dataset size. In practice, however, the performance of the protocol is greatly affected by the total number of content blocks reported by monitors. Our system aims to find signatures of worms that propagate quickly. Thus, performance is as important as privacy protection. With the relaxed notion of privacy, we achieve the performance required for effective worm signature detection.

Kissner and Song [70] provide another protocol that protects privacy and correctness when malicious adversaries are present. Malicious adversaries may behave arbitrarily, in contrast to hones-but-curious adversaries. The protocol provides such strong privacy by using commitments on datasets before proceeding with the main protocol, and by attaching

a zero-knowledge proof to every exchanged message. That requires additional communication and computation. Unfortunately, this protocol is too expensive to be used for our purposes.

8.2 Preliminaries

The privacy-preserving set operation framework proposed by Kissner and Song [70] uses a semantically secure, additively homomorphic public-key cryptosystem. We describe the threshold homomorphic cryptosystem and then briefly introduce the privacy-preserving set operation framework in this section.

8.2.1 Threshold Homomorphic Cryptosystem

Let $E_{pk}(\cdot)$ denote the encryption function with public key pk . The cryptosystem supports the following operations without knowledge of the private key sk:

1. $E_{pk}(m_1 + m_2) = E_{pk}(m_1) +_h E_{pk}(m_2)$.
2. $E_{pk}(c \times m_1) = c *_h E_{pk}(m_1)$.

The $+_h$ and $*_h$ represent the homomorphic addition and the homomorphic multiplication operations, respectively. When such operations are performed, the resulting ciphertexts should be re-randomized for security. In re-randomization, a ciphertext is transformed so as to form an encryption of the same plaintext, under a different random string than the one originally used.

The privacy-preserving multiset operation framework also requires that the private key in the cryptosystem is shared by more than $(c + 1)$ nodes so that any coalition of at most c nodes cannot decrypt messages exchanged during the protocol execution. We employ the threshold Paillier cryptosystem [50, 51] that extends the homomorphic Paillier cryptosystem [100].

In the $(c + 1, M)$-threshold Paillier cryptosystem, where M is the number of participating monitors in the system, and c is the maximum number of malicious monitors the system can tolerate, messages are encrypted using a public key pk, and the corresponding private key sk is shared by a group of M participants. At least $c + 1$ of these participants can efficiently decrypt encrypted messages, while no coalition of less than $c + 1$ participants can learn useful information from encrypted messages.

Let $\Delta = M!$. The key generation, encryption, and decryption processes of the $(c+1, M)$-threshold Paillier cryptosystem are performed as the follows:

- **Key Generation:** choose two safe primes [1] $p = 2p' + 1$ and $q = 2q' + 1$ where p' and q' are also large random prime. Set $m = p'q'$ and $n = pq$. Let g be a generator and $\beta \in Z_n^*$ is a random element. The public key is $\mathsf{pk} = (n, g)$. The secret key $\mathsf{sk} = \beta \times m$ is shared using the Shamir scheme [116] modulo mn.
- **Encryption:** For a message P, $E_{pk}(P) = g^P x^n \mod n^2$.
- **Re-randomization:** To re-randomize a cipher text C, multiply it by a random encryption of 0, *i.e.*, compute $Cr^n \mod n^2$ for $r \in Z_n^*$.
- **Partial Decryption:** For a cipher text C, the i-th share is computed by $C_i = Dec_{\mathsf{sk}_i}(C) = C^{2\Delta \mathsf{sk}_i} \mod n^2$.
- **Recovery:** Give a set of $c+1$ valid shares, compute the plain text using the Lagrange interpolation on the exponents.

Note that, in the original scheme [50], Δ was defined to be $M!$ assuming $(c+1, M)$-threshold decryption so that no modular root extraction is required in the *Recovery* operation. In order to avoid the prohibitively large computation overhead when many monitors are in the system, we group all monitors into $c+1$ groups, and all monitors in the same group hold the same share of a secret key, sk_i, *i.e.*, node j holds $\mathsf{sk}_{j \mod c+1}$. In this way, we can limit Δ to be $(c+1)!$ ($c \ll M$) while still avoiding modular root extraction. Because there are at most c malicious nodes, malicious nodes cannot acquire all $c+1$ secret keys.

8.2.2 Privacy-Preserving Set Operations

Kissner and Song [70] showed that various privacy-preserving operations on multisets can be performed using polynomial representations and employing the mathematical properties of polynomials.

Let Z_n be a sufficiently large plaintext domain and $h(\cdot)$ denote a cryptographic hash function that maps an element a into a domain $\mathbb{Z} \subset Z_n$. When $a \neq a'$, there is only a negligible probability that $h(a) = h(a')$. Given a multiset $A = \{a_j\}_{1 \leq j \leq k}$, A can be represented with a polynomial $f \in Z_n[x]$ whose roots are collision-resistant hash values of elements in the multiset A [52, 70]:

[1] A safe prime is a prime number of the form $2p + 1$, where p is also a prime. For example, 5, 7, and 11 are safe primes.

$$f(x) = \prod_{a \in A}(x - h(a))$$

The polynomial representation of a multiset enables various operations on multisets where an element a can appear multiple times in a set and $freq_A(a)$ denotes the multiplicity of a in the multiset A. We present an important lemma that is the basis of the THRESHOLD-SET-UNION-HBC protocol design. The proof is provided in [70].

Lemma 1. *When f and g represent sets A and B, respectively, $f^{(d)}$ denotes d-th derivative of the polynomial f, and, $r \in Z_n[x]$ and $s \in Z_n[x]$ are randomly chosen polynomials:*

$$\begin{aligned}
a \in A &\Leftrightarrow (x - h(a)) \mid f \\
a \in A \cup B &\Leftrightarrow (x - h(a)) \mid f \times g \\
a \in A \cap B &\Leftrightarrow (x - h(a)) \mid f \times r + g \times s \\
freq_A(a) > d &\Leftrightarrow (x - h(a)) \mid f \times r + (f^{(1)} + \cdots + f^{(d)}) \times s \\
freq_A(a) \leq d &\Leftrightarrow (x - a) \nmid f \times r + (f^{(1)} + \cdots + f^{(d)}) \times s
\end{aligned}$$

The lemma shows that we can perform the basic operations such as membership test, union, and intersection on the multisets represented by the polynomials. Moreover, the polynomial representation allows to test whether the multiplicity of an element in the represented multiset is larger than a certain number or not.

The encryption of polynomial f is computed by encrypting all the coefficients of f.

$$E_{pk}(f) \text{ is } E_{pk}(f[0]) *_h x^i +_h \cdots +_h E_{pk}(f[deg(f)]) *_h x^0$$

where $f[i]$ denotes the i-th coefficient of f.

The framework supports the following operations on the encrypted polynomials f_1 and f_2 without knowing the secret key, thank to the homomorphic property of Paillier's cryptosystem. We use $deg(f_1)$ and $deg(f_2)$ to denote the degrees of f_1 and f_2, respectively.

- **Sum of encrypted polynomials:** given the encryptions of the polynomial f_1 and f_2, we can efficiently compute the encryption of the polynomial $g = f_1 + f_2$, by calculating $E_{pk}(g[i]) = E_{pk}(f_1[i]) +_h E_{pk}(f_2[i])$, $(0 \leq i \leq max[deg(f_1), deg(f_2)])$.
- **Product of an unencrypted polynomial and an encrypted polynomial:** given a polynomial f_2 and the encryption of polynomial f_1, we can efficiently compute the encryption of polynomial $g = f_1 \times f_2$, by calculating the encryption of each coefficient $E_{pk}(g[i]) = (f_2[0] *_h E_{pk}(f_1[i])) +_h (f_2[1] *_h E_{pk}(()f_1[i-1])) +_h \cdots +_h (f_2[i] *_h E_{pk}(f_1[0]))$, $(0 \leq i \leq deg(f_1) + deg(f_2))$.

- **Derivative of an encrypted polynomial:** given the encryption of polynomial f_1, we can efficiently compute the encryption of polynomial $g = \frac{d}{dx} f_1$, by calculating the encryption of each coefficient $E_{pk}(g[i]) = (i+1) *_h E_{pk}(f_1[i+1])$, $(0 \leq i \leq deg(f_1) - 1)$.
- **Evaluation of an encrypted polynomial at an unencrypted point:** given the encryption of polynomial f_1, we can efficiently compute the encryption of $y = f_1(x)$, by calculating $E_{pk}(y) = (x^0 *_h E_{pk}(f_1[0])) +_h (x^1 *_h E_{pk}(f_1[1])) +_h \cdots +_h (x^{deg(f_1)} *_h E_{pk}(f_1[deg(f_1)]))$.

Utilizing the above operations on encrypted polynomials, we can securely compute results according to the multiset operations described in Lemma 1 without the trusted third party.

8.3 Protocols

We now describe protocols used for identifying content blocks that are reported by more than λ_G distributed monitors. We first present the basic protocol, an adoption of THRESHOLD-SET-UNION-HBC protocol proposed by Kissner and Song [70]. Then, we extend the protocol further to deal with high computational overhead when the total number of reported content blocks is large (Section 8.3.2). We use an approximate counting scheme to reduce the overhead of the THRESHOLD-SET-UNION-HBC protocol further (Section 8.3.3).

8.3.1 Basic Protocol

Figure 8.1 describes the modified THRESHOLD-SET-UNION-HBC protocol. When a monitor publishes a content block P, we use anonymous routing [24, 22, 111, 112, 84] to hide the origin of the content block. Assuming that the additively homomorphic threshold cryptosystem is semantically secure and re-randomization technique is used in the collection phase, with overwhelming probability, any coalition of at most c probablistic polynomial time-bounded (PPT) honest-but-curious adversaries cannot recover items that appear less than λ_G times. The detailed analysis of the privacy and the overhead of this protocol is provided by Kissner and Song [70].

8.3.2 Extended Protocol for Reducing Computational Cost

The most expensive, but most frequently performed operation in the basic protocol is the multiplication of an encrypted polynomial and a plain polynomial. (See the Collection and

the Collaborative Reduction Phases in Figure 8.1.) For two polynomials of degree d_1 and d_2, the multiplication operation results in $O(d_1 \cdot d_2)$ homomorphic multiplication, which is akin to modular exponentiation of big numbers. The degree of the final polynomial from the collection phase is linear to the total number of content blocks reported by all monitors. Then, we perform homomorphic multiplication operations multiple times on the polynomial. Thus, the computational cost of the protocol increases quadratically to the total number of content blocks.

We reduce the overhead by using multiple low-degree polynomials rather than using a high-degree polynomial. All monitors agree on some hash function h' that maps payload pattern strings into B bins; generate a polynomial for each bin. Then, they run the basic protocol with only the polynomial from the first bin, and repeat the procedure for each bin. After B runs of the basic protocol, the computation completes. Because every node uses the same hash function, the correctness of the basic protocol still holds.

Assuming h' uniformly distributes content blocks into B bins, this technique reduces the computational overhead by a factor of B. We can achieve better performance by executing the B runs concurrently and utilizing the computational power of monitors that are idle. Only $c + 1$ nodes participate in the collaborative reduction phase and the first step of decryption phase. All other monitors are idle.

However, using a large B is undesirable. The use of B reduces the size of the original domain by a factor of B and gives adversaries more information about the values of content blocks. For example, when the degree of a polynomial for the jth bin has increased by two after a monitor M_i multiplies its polynomial during the collection phase, an adversary can infer that M_i has two content blocks that fall into jth bin.

Each node may add padding to hide the (non)existence of items whose hash values fall into the bin. But this increases the degree of the polynomials, and cancels out the performance gain from our extended protocol. Thus, in our work, we avoid item padding, assuming that the privacy protection with the B-times smaller domain is still acceptable.

Choice of the hash function h' is important for the effectiveness of this extended protocol. If the hash function fails to evenly distribute the content blocks, the performance benefit from the protocol is small. Much effort has been made to develop collision-resistant hash functions by using randomization [14, 12]. However, they are not directly applicable to our problem because, in our problem setting, all monitors must agree on the same hash function. The same content blocks must be hashed to the same value in order to accurately check if the number of a content block exceeds the threshold λ_G . We expect that similar problems would arise in many distributed systems. Developing hash functions suitable for

this situation would be an interesting problem to be researched.

8.3.3 Approximate Payload Pattern Counting

When the frequency of a worm payload pattern across monitors is much larger than that of non-worm content blocks, we can take an approximate counting approach. Basically, each monitor selects a content block in its private set S_i with a probability ρ and computes its polynomial based on only those selected content blocks. This reduces the degree of the final polynomial by a factor of ρ^{-1} and the number of computations by a factor of ρ^{-2}. The new frequency threshold then becomes $\lambda_G{}' = \lambda_G \cdot \rho$. The protocol computes the content blocks that appear in more than $\lambda_G{}'$ monitors sets. When the actual frequency $F(P)$ of a content block P is y, the probability of identifying P is $1 - \sum_{x=0}^{\lambda'_G} \binom{y}{x} \rho^x (1-\rho^{y-x})$.

8.4 Implementation

We now describe our prototype of the distributed signature generation system, which uses the privacy-preserving THRESHOLD-SET-UNION-HBC protocol described in the previous section. All monitors are connected with an overlay network that supports efficient multicast [26, 18]. With the multicast facility, a message can be delivered to all M monitors after $\log M$ hops in the overlay network. Autograph is extended to have a TSU (Threshold-Set-Union) module attached to the signature generation step.

Autograph: Autograph is responsible for selecting suspicious flows from monitored traffic, and performing the content-prevalence analysis to choose suspicious content blocks, whose prevalence measure $f(P)$ exceeds a local threshold θ_G . Those content blocks are passed to the TSU module. In this work, we use $\theta_G = 2$ for a monitor in the network of a 2^{13} IP address space.

TSU module: This module takes the content blocks discovered by Autograph, and participates in the distributed THRESHOLD-SET-UNION-HBC protocol. The TSU module maintains the private key sk and a public key pk, distributed by a trusted group manager. We use the threshold Paillier cryptosystem [50] and the THRESHOLD-SET-UNION-HBC protocol atop the cryptographic primitives included in the SFS toolkit [81].

For each measurement interval, TSU assigns each content block to one of B bins by applying a hash function agreed on with the other monitors, samples payload pattern strings with a probability ρ, and generates B polynomials each of which represents the payload

pattern strings in each bin. A TSU module participates in B THRESHOLD-SET-UNION-HBC instances concurrently.

Forcing all monitors to participate in the collection phase, computing the product of all the polynomials along the ring of monitors, is not efficient. In practice, many monitors may not have content blocks to report. Their participation in the protocol does not affect the final results but adds more overhead. In our implementation, a monitor M_i sends a short PING message to the next monitor M_{i+1} before sending the product of polynomials. If monitor M_{i+1} has content blocks to share, it replies back with a PONG YES message, and the monitor M_i sends the product of polynomials. If monitor M_{i+1} responses with a PONG NO message, M_i contacts the next monitor M_{i+2}, and so on. This process continues until the monitor M_i finds a monitor with content blocks or contacts the monitor that initiated the collection phase.

8.5 Experimental Results

We deployed a network of TSU modules over the Emulab [142] and measured the wall-clock time between receiving content blocks from Autograph and completing a round of THRESHOLD-SET-UNION-HBC computation. This time T_o is the 'lag' due to distributed monitoring, mentioned in Chapter 6. If the lag is too large, the system is not useful for worm signature detection. In the evaluation of worm signature detection time in Chapter 6, we assumed $T_o \leq 5$ minutes. In this section, we examine if we can complete the computation within a reasonable time such as $T_o \leq 5$ minutes.

TSU makes a heavy use of the threshold-homomorphic-cryptosytem operations. Table 8.1 shows the speed of threshold Paillier operations given as the mean over 100 runs across different keys. We vary the threshold c of the cryptosystem. We measured the speed of operations both on computers running in 64-bit mode and 32-bit mode. The computer running in 64-bit mode has a 3.6GHz Xeon processor. All other benchmarks and experiments presented in this section are performed on 3GHz Intel Pentium 4 processor-based computers running in 32-bit mode. The homomorphic multiplication is expensive. The decryption operation includes the partial decryption and the recovery operations, so the speed of the decryption operation is linear to the threshold c. Because the decryption operation is the most expensive operation, the extended protocol for reducing network delay (Appendix A), that requires decryption of polynomials several times in the middle of the protocol execution, is not suitable for our problem setting. The use of 64-bit mode processors improves the operation speed significantly. Considering the speed difference of each operation, we ex-

						3.6GHz Xeon in 64-bit mode
	c=1	c=3	c=7	c=15	c=31	c=64
$+_h$	0.014	0.015	0.014	0.015	0.015	0.015
$*_h$	6.2	7.2	6.5	6.2	5.8	5.1
$E_{pk}(.)$	19.9	20.4	20.1	21.2	22.1	24.3
$D_{\mathsf{sk}}(.)$	15.5	33.1	65.3	148.0	352.6	977.5
						3.0GHz Pentium 4 in 32-bit mode
	c=1	c=3	c=7	c=15	c=31	c=64
$+_h$	0.038	0.036	0.038	0.038	0.038	0.038
$*_h$	15.7	16.6	14.8	17.9	17.4	17.5
$E_{pk}(.)$	55.9	55.4	56.5	60.1	62.4	69.3
$D_{\mathsf{sk}}(.)$	46.2	90.4	192.3	416.3	993.2	2754.6

Table 8.1: Speed (ms) of operations of 1024-bit threshold Paillier cryptosystem

pect that the use of 64-bit mode processors would reduce the T_o measured in the remaining experiments by a factor of $4 - 9$.

We performed experiments on Emulab varying the number of monitors M, the number of maximum attackers c, and the average number of payload pattern strings per monitor $\mathrm{E}\left[|S_i|\right]$. We use 1024 bit public keys. The average link latency between monitors is 49 msec throughout the experiments.

Figure 8.2 (a) shows the protocol running time in seconds varying the number of monitors. We also vary the average number of content blocks reported by each monitor, $E[S_i]$. It turns out the running time is $O((M \cdot \mathrm{E}\left[|S_i|\right])^2)$ and grows quadratically as the number of monitors increases. However, the results also shows that it is feasible to finish the computation if we can manage to limit the average number of reported payload patterns to be small. For example, 128 monitors can complete the computation within 5 minutes if the average number of payloads reported in each THRESHOLD-SET-UNION-HBC protocol is smaller than 1.

Figure 8.2 (b) shows the 'lag' T_o varying the number of honest-but-curious attackers (c). The lag increases as the number of tolerable attackers increases. There is always a trade-off between privacy protection and efficiency.

We then use a traffic trace from a campus network to estimate the time to deal with content blocks generated from legitimate traffic. In our experiment, each content block is mapped to one of 10 polynomials and each monitor participates in 10 instances of THRESHOLD-SET-UNION-HBC computation process concurrently. With the traffic monitoring module's threshold $\theta_G = 2$, the number of payload patterns each monitor generates from the traffic trace for an hour long time window varies between 0 and hundreds (`smtp`), but mostly below 10. With a sampling rate $\rho = 0.3$, the average number of payload patterns reported by a monitor is 0.3 which is reasonably small. When a worm propagates, most of the monitors will observe the worm payload patterns. With $\rho = 0.3$, only 3 out of 10 monitors will report

the payload patterns.

From the experiments, we conclude that privacy-preserving payload sharing with the extended THRESHOLD-SET-UNION-HBC protocol is feasible in a distributed signature generation system with a hundred monitors.

8.6 Summary

In this chapter, we presented a privacy-preserving distributed signature generation system that is one of the first applications that apply the THRESHOLD-SET-UNION-HBC protocol. This system provides cryptographically provable privacy protection, but is computationally expensive and has limited scalability. The system presented in this chapter is suitable for distributed signature generation where less than hundreds of participants collaborate. For example, signature generation monitors from different autonomous domains can choose to exchange locally discovered candidate signatures in order to gain reasonable confidence.

Our original protocol takes the polynomial representation of each private set, encrypts each coefficient of the polynomial with homomorphic encryption, and computes the product of the polynomial representations from participants. Differentiation of the polynomial cancels out the item that does not appear in more than λ participants. The original provides strong privacy protection. However, the operations of homomorphic multiplications and decryption are prohibitively expensive.

In Section 8.3, we extended the protocol to reduce the computational cost and combine an approximate counting approach to reduce the computational cost further. The extended protocol, however, provides weak privacy-preservation compared to the original protocol, and approximate counting may result false negatives. The experimental results with our non-optimized implementation show that currently we can complete the global signature generation process within 5 minutes if not more than hundreds monitors participate and an aggressive approximate counting is employed. However, note that with 64 bit mode processors, we could reduce the basic operation speeds significantly. The use of cryptographic accelerators is also promising for reducing the time caused by many big number exponentiation operations. More interestingly, most of the computation involved in the protocol can be easily parallelized; with the advance in multi-core processors [32], we expect our privacy-preserving signature generation system can scale to more than hundreds of monitors easily.

Protocol: modified-THRESHOLD-SET-UNION-HBC

Input: There are $M \geq 2$ honest-but-curious monitors, c ($< M$) monitors dishonestly colluding, each with a private input of payload pattern strings S_i. The monitors share the secret key sk, to which pkis the corresponding public key for a threshold homomorphic cryptosystem. F_j ($1 \leq j < t$) are fixed polynomials of degree j which have no common factors or roots representing any payload pattern string.
Output: Each player learns the payload patterns in $S_i \cap R$ where R has payload pattern strings found in more than t monitors' private input sets.

(Initialization Phase) Each monitor M_i ($0 \leq i < M$) calculates the polynomial f_i representing S_i.
(Collection Phase) All monitors collaboratively compute $E_{pk}(\prod_{i=0}^{M-1} f_i)$ as follows:

1. M_0 sends the encryption of the polynomial $\lambda_0 = E_{pk}(f_0)$ to M_1.
2. Each monitor M_i ($i = 1, 2, ..., M-1$)
   (a) receives the encryption of the polynomial λ_{i-1} from M_{i-1}.
   (b) computes $\lambda_i = \lambda_{i-1} * f_i$.
   (c) sends λ_i to the next monitor $M_{i+1 \mod M}$.
3. M_0 receives the encryption of the polynomial $p = \lambda_{M-1}$ from M_{M-1}.

(Collaborative Reduction Phase) $c+1$ monitors collaboratively compute the encryption of the polynomial representing R.

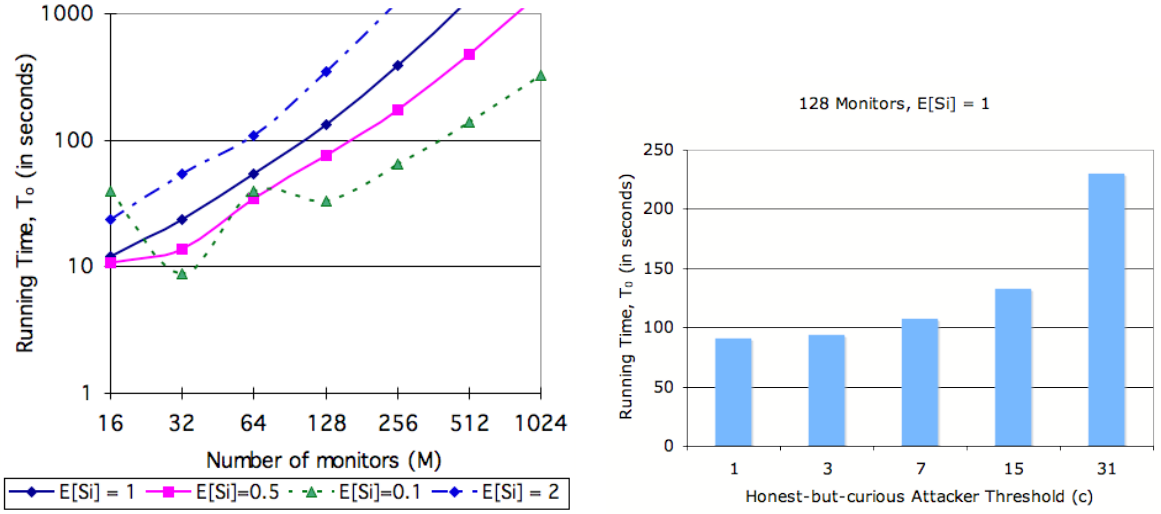
1. M_0 distributes the encryption of the polynomial p to M_1, M_2, .. , M_c.
2. Each monitor M_i ($i = 0, 1, .., c$)
   (a) computes the encryption of the 1, .. , tst derivatives of p, denoted $p^{(1)}, ..., p^t$.
   (b) chooses random polynomials $r_{i,1}, ..., r_{i,t}$ of the same degree as p.
   (c) computes the encryption of the polynomial $\sum_{j=0}^{t} p^{(j)} * F_j * r_{i,j}$.
   (d) sends the result to c monitors M_i ($i = 0, 1, .., c$).
3. $c+1$ monitors M_i ($i = 0, 1, .., c$) compute the encryption of the polynomial $\Phi = \sum_{j=0}^{t} p^{(j)} * F_j * (\sum_{k=0}^{c} r_{k,j})$.

(Decryption Phase) All M monitors acquire the polynomial representation of R.

1. Each monitor M_i ($i = 0, 1, .., c$) sends the partial decryption to all M monitors.
2. All M monitors M_i ($i = 0, 1, .., M-1$) perform the recovery operation based on the $c+1$ partial decryption.
3. All M monitors have Φ that represents R.

(Evaluation Phase) All M monitors evaluate $\Phi(P)$ for all $P \in S_i$. If $\Phi(P) = 0$, $P \in R \cap S_i$.
(Publication Phase) All M monitors sign P s.t. $P \in R \cap S_i$ with a group signature and send via anonymous routing.
(Verification Phase) Each monitor can verify if $P \in R$ by evaluating $\Phi(P) == 0$ and checking its signature.

Figure 8.1: **Modified** THRESHOLD-SET-UNION-HBC **Protocol:** secure against honest-but-curious adversaries



(a) (b)

Figure 8.2: **Running time for one** THRESHOLD-SET-UNION-HBC **computation:** (a) The running time increases quadratically to the number of monitors M and the average set size $\mathrm{E}[S_i]$. (b) Running time vs. c (a threshold cryptosystem parameter). $\mathrm{E}[S_i] = 1$. M=128.

Chapter 9

Related Work

9.1 Automated Worm Signature Generation

Singh *et al.* [119] generate signatures for novel worms by measuring packet content prevalence and address dispersion at a single monitoring point. Their system, EarlyBird, avoids the computational cost of flow reassembly, but is susceptible to attacks that spread worm-specific byte patterns over a sequence of short packets. Autograph instead incurs the expense of flow reassembly, but mitigates that expense by *first* identifying suspicious flows, and *thereafter* performing flow reassembly and content analysis only on those flows. EarlyBird reverses these stages; it finds sub-packet content strings first, and applies techniques to filter out innocuous content strings second. Autograph and EarlyBird both make use of Rabin fingerprints, though in different ways: Autograph's COPP technique uses them as did LBFS, to break flow payloads into non-overlapping, variable-length chunks efficiently, based on payload content. EarlyBird uses them to generate hashes of overlapping, fixed-length chunks at every byte offset in a packet efficiently. Singh *et al.* independently describe using a white-list to disallow signatures that cause false positives (described herein as a blacklist for signatures, rather than a white-list for traffic), and report examples of false positives that are prevented with such a white-list [119].

Provos [105] observes the complementary nature of honeypots and content-based signature generation; he suggests providing payloads gathered by `honeyd` to automated signature generation systems such as Honeycomb [73]. DACODA [35] is one example that forwards payload information captured from honeypot to a payload analyzer where the network payload information is correlated with control flow hijacking activities. Autograph would similarly benefit from `honeyd`'s captured payloads. Furthermore, if `honeyd` partic-

ipated in tattler, Autograph's detection of suspicious IP addresses would be sped, with less communication than that required to transfer complete captured payloads from instances of `honeyd` to instances of Autograph.

Kreibich and Crowcroft [73] describe Honeycomb, a system that gathers suspicious traffic using a honeypot, and searches for least common substrings in that traffic to generate worm signatures. Honeycomb relies on the inherent suspiciousness of traffic received by a honeypot to limit the traffic considered for signature generation to truly suspicious flows. This approach to gathering suspicious traffic is complementary to that adopted in Autograph; we need investigate acquiring suspicious flows using honeypots for signature generation by Autograph in future. The evaluation of Honeycomb assumes all traffic received by a honeypot is suspicious; that assumption may not always hold, in particular if attackers deliberately submit innocuous traffic to the system. Autograph, Honeycomb, and EarlyBird will face that threat as knowledge of their deployment spreads. We combat the threat by vetting candidate signatures for false positives among many distributed monitors.

Our work is the first we know to evaluate the trade-off between earliness of detection of a novel worm and generation of signatures that cause false positives in content-based signature detection. Since Autograph, Honeycomb, and EarlyBird demonstrated their promising results of automated signature generation, much research has been published exploring faster and more robust automated worm signature generation.

Wang *et al.* [139] extended their prior work [138] on payload-based network intrusion detection to generate Internet worm signatures. Their system, PAYL, detects anomalous packets based on the model of normal payload bytes distribution and extract the invariant portion of packet content across similarly anomalous packets using the longest common subsequence algorithm. We expect their heuristic to detect anomalous packets can be used to improve our suspicious flow classifier. Karamcheti *et al.* [67] proposed a malicious network traffic detection algorithm by utilizing the payload content similarity of worm traffic. They model the legitimate traffic with the inverse distributions of substrings in a packet, and any deviation from the model indicates on-going worm activities. However, their work does not target to distinguish suspicious flows nor generate signature useful for current content-based filtering.

Cai *et al.* [16] proposed a collaborative worm detection system, WormShield. WormShield aggregates the statistics of payload pattern fingerprint and address dispersion in an overlay network; detects anomalous payload pattern fingerprints based on the aggregate statistics. Their results corroborate our analysis on the benefit of distributed monitoring. They protect privacy by using SHA-1 hash instead of the actual payload substrings in the aggregation.

However, the privacy protection does not prevent dictionary attacks on private data.

Autograph is not able to handle maximally-varying polymorphic worms, but Polygraph [93] demonstrated that it is possible to generate accurate signatures for polymorphic worms, because there are some features that must be present in worm infection attempts to successfully exploit the target machine. Polygraph showed automatic signature-generation techniques that extract a set of small substrings from suspicious flow pool, taking a machine-learning approach. Hamsa [76] is another learning based automated signature generation system for polymorphic worms. By employing more efficient data structures and an improved model in analyzing the invariant content among suspicious flows, Hamsa achieves significant improvements in signature generation speed, accuracy, and attack resilience over Polygraph. Paragraph [94] demonstrated attacks against such learning-based signature generation systems in an adversarial environment, so the pattern-extraction-based signature generation for polymorphic worms is still a problem to be addressed.

Another approach to automate signature generation is based on application and exploit semantic information collected from a host [95, 34, 77, 15]. This host-based approach can provide more robust signature generation against polymorphic worms, and generate more expressive forms of signatures than network-based pattern matching signatures generated by Autograph. However, the network-based pattern matching signature still holds its own value: signature matching is fast so can be deployed in the network to stop worms before reaching hosts.

9.2 Distributed Monitoring

Distributed monitoring is not a novel concept in network security. For the last decade, much effort has been made to take advantage of distributed monitoring to detect network intrusions that are difficult to detect from a single monitoring point. DIDS [120] is an early example of distributed intrusion detection system that collects security-relevant event from a number of monitors on hosts to track of users as they change account name moving around the networks. Later, many systems such as EMERALD [104], AAFID [46], GrIDS [129], and Argus [65] present various techniques to aggregate observation from a large number of distributed monitors in a more scalable and reliable fashion.

As more and more computers are connected to networks and the risk of wide-spread, large-scale attacks grows, distributed monitoring becomes one of the important components in Today's intrusion detection systems. Snort [131] allows a multi-tier deployment to provide scalability, security, and performance. The sensors interpret intrusion data gath-

ered from lower tier sensors distributed across an enterprise and forward interesting data to higher tier sensors [72]. Bro [102] also offers the option to exchange state information with other Bro peers to detect distributed attacks [121, 74]. While the benefit of distributed monitoring is already widely accepted in network security, our contribution in this thesis is to adopt distributed monitoring in automated worm signature generation, which is challenging because signature generation involves privacy-sensitive information exchanges across different domains.

Yegneswaran *et al.* [146] corroborate the benefit of distributed monitoring, both in speeding the accurate accumulation of port scanners' source IP addresses, and in speeding the accurate determination of port scanning volume. Their DOMINO system detects port scanners using active-sinks (honeypots), both to generate source IP address blacklists for use in address-based traffic filtering, and to detect an increase in port scanning activity on a port with high confidence. The evaluation of DOMINO focuses on speed and accuracy in determining port scan volume and port scanners' IP addresses, whereas our evaluation of Autograph focuses on speed and accuracy in generating worm signatures, as influenced by the speed and accuracy of worm payload accumulation.

Nojiri *et al.* [96] describe models for peer-to-peer- based attack mitigation strategies and investigate the efficacy of those strategies in stopping large scale Internet worms. In their peer-to-peer based strategies, members share the observed attack reports with their cooperating friend members. Each member can decide whether it uses the reports from friend members or shares with other friend members, based on its local policy. Their simulation results show that, in general, a large number of cooperating members does better in suppressing worms but is much worse in dealing with false alarms. They also show that we need different strategies based on the propagation behavior of worms. In our work, we proposed two different distributed payload sharing mechanisms for worms that propagate using localized scanning strategies, and for worms that propagate using random scanning strategies. Unlike the systems presented in their work that share only attack reports confirmed locally, our payload sharing network requires information to be shared before it is not locally confirmed as part of attacks. Thus, the focus of our work is how to protect the privacy in the shared information while enabling information sharing.

9.3 Privacy-Preserving Hot Item Identification

We adopt privacy-preserving set operations [70] to find payload patterns that appear in multiple monitors. The protocol based on privacy-preserving set operation is secure un-

der a very strict notion of privacy [55], but is slow if we apply it directly to the signature generation problem. Thus, we combined it with approximate counting in our implementation of the protocol. Our work is the first practical system that adopts the framework. The HOTITEM-ID protocol [69] provides more efficient and flexible results by being designed to achieve more restricted notions of privacy.

Even though many security applications have adopted the distributed monitoring approach to improve their attack detection performance, little effort has been made to address the privacy-related issues. Most work either assumed a trusted environment, or a trusted central server that aggregate reports from distributed monitors [41]. Lincoln *et al.* [78] describe privacy-breaching attacks against distributed network monitoring node. Their approach, however, still relies on a trusted central servers and avoids privacy-sensitive data such as payload information. Wormshield [17] attempts to protect the privacy of data by using hash values instead of payload data. However, the approach does not prevent dictionary attacks. Our work that addresses the privacy issues in a distributed fashion with more restricted notion of privacy.

Allman *et al.* [3] propose an architecture for using cross-organization information sharing to understand distributed, coordinated attacks. In their framework, a small number of "detectives" systems such as honeypots leverage existing wide-scale generic traffic monitors as "witnesses" of malicious network events the detectives are investigating. They use a private matching mechanism to allow for detectives to question witnesses without revealing the detail of the attacks under investigation or witnesses' private information unrelated to the attacks. While our work focuses on sharing payloads for faster signature generations and does not rely on the specialized monitors to initiate information sharing, our work shares much similarity with their proposed system. For instance, both systems utilize intelligent intrusion detection systems such as honeypots (but in different ways), protect private information, and prevent cheating by relying on multiple monitors reporting mutually observed network events.

Privacy-preserving collection of statistics about computer configurations has also been considered in previous work [137, 61]. Like the work in [78], they do not give a concrete definition of security, but instead a technique for heuristically confusing attackers. Their approach also relies on chains of trust between friends and the collection of statistics is gathered by querying a subset of distributed nodes.

Chapter 10

Conclusions

In this dissertation, we developed an automated Internet worm signature generation system and the techniques to speed its signature generation with information sharing among distributed monitors while protecting privacy. In concluding this dissertation, this chapter first revisits the desired requirement of automated worm signature generation. Key contributions are then discussed, after which important lessons learned about automated signature generation will be presented.

10.1 Desired Requirements Revisited

The system and techniques presented in this dissertation are designed to meet the desired requirements of automated signature generation for signature-based worm traffic detection.

- **Signature Quality:** Autograph with COPP and content prevalence analysis generates specific and selective signatures for an Internet worm as long as there are unique byte sequences that can be used for worm's payload signature. Signatures for polymorphic or metamorphic worms are difficult to generate using the current Autograph. Polygraph [93] and many other techniques, proposed after Autograph, shed light on more accurate automated signature generation even for polymorphic or metamorphic worms.

- **Early Signature Generation:** By making multiple monitors dispersed in the Internet collaborate, our systems generate signatures for worms that propagate by scanning the Internet address space before they spread wild. The collaboration happens in the three different stages of our systems. First, the distributed monitors collaborate to

identify the sources of suspicious flows. Second, the distributed monitors collaborate to share the identified suspicious payload pattern strings. Third, the monitors share generated signatures. This type of collaboration makes our system viable even when a sigle monitor is not sufficient enough to collect enough samples required for signature generation.

- **Minimal Real-time Operator Intervention:** Our systems allow all the signature generation process to be performed without human intervention.

- **Robust against Attacks:** Distributed payload sharing and signature generation can be considered 'voting' among multiple monitors, we avoid generating false signatures due to attacks against a small number of monitors. We preserve the owner and data privacy of participating networks and shared data, so that any curious attackers cannot use our distributed systems to gain private information about participating networks.

- **Minimizing the Effect on Legitimate Internet Use:** Our system relies on passive network monitoring. If the monitors are installed at the boundary of each monitored network, users in the monitored networks does not experience any additional overhead or inconvenience such as installing software or changing the configuration of personal computers. Generating false signatures can disrupt legitimate traffic and thus must be avoided by all means. Our analysis with real network traces showed the false positive rate is low, and the system generates only a handful number of signatures. We still need more investigation with more data sets. However, considering that we observed low false positive rates in the analysis with traces collected from multiple, independent places at different time period, the result is still promising.

- **Preserving Privacy:** We provide two privacy-preserving payload sharing techniques.

While the results of our analysis showed our proposed techniques have promise in the automated worm signature generation, we stress that it is yet too early for us to conclude that our techniques are the perfect solution for the problem. Our experiments and analysis are based only on a few traces, and focused only on worms that propagate by scanning IP addresses. Further investigation with more traffic traces from different networks against a wider variety of Internet worms is still required. We believe, however, that the techniques and the evaluation methodology discussed in this thesis shed new light on worm detection and prevention technology, and raise the bar in future worm design.

10.2 Contributions

We catalog the contribution of this work:

- **Automated Signature Generation Method.** We proposed and evaluated a technique that automates pattern-based signature generation by utilizing content prevalence analysis. This technique is effective in detecting worms that possess the following properties: 1) content invariance, 2) voluminous traffic, and 3) many-to-many communication. With our Autograph system, we presented these properties are a good basis to distinguish such worm traffic from other legitimate traffic. Furthermore, the invariant portion of worm payloads can be extracted by a simple prevalence analysis and used as a signature for the worm. We provide the criteria to evaluate automated signature generation systems. Inspired by our early work in Autograph development and contemporaneous work such as Honeycomb [73] and EarlyBird [119], many different techniques for automated signature generation have been proposed from the security research community since our work.

- **Distributed Monitoring as a Mean of Fast Signature Generation.** This work has provided the basis for distributed monitoring for fast signature generation for worms that propagate in moderate speed, and thus the number of worm payload samples a single monitor can accumulate is limited [1] The *tattler* mechanism provides a bandwidth-efficient way to gossip scanners' IP addresses and we showed the value of distributed scanner detection in terms of signature generation speed. We provided techniques for suspicious payload information useful in the early detection of worm signatures. We demonstrated the efficient ways to mine a useful piece of information out of huge amount of traffic data. We also evaluated the benefit of payload sharing in automated signature generation qualitatively and quantitatively.

- **Privacy-Preserving Information Sharing.** Recognizing the privacy issues and attacks in the concept of payload sharing, this work provides the technique to share payload while protecting the privacy of participating monitors. Most prior work in distributed monitoring for network security has avoided the privacy issues or has employed a trusted, centralized system to aggregate information from distributed monitors. This work showed that fully distributed, privacy-preserving information sharing is feasible

[1] When a worm propagates as fast as a *Warhol* worm or a *flash* worm [127, 140] and targets a large fraction of hosts in the Internet, we expect that a single monitor can capture enough payload samples to generate the signatures and deploy them locally.

by applying probabilistic and cryptographically proved privacy-preserving techniques to automated signature generation problem, which requires fast information exchange.

10.3 Lessons Learned

As touched on in the contributions above, the analysis of our signature generation techniques illuminated a number of important elements in automated signature generation system development.

- **Quality of Autograph-generated Signatures.** The suspicious flow pool constructed with a simple port-scanner-based suspicious flow selection is dominated by present scanning worm flows. The content blocks of worms' invariant portions are topped for the prevalence histogram of content blocks from suspicious flows. (Section 3.2) Iterative selection of prevalent content blocks with carefully chosen parameters generates a selective set of signatures that catch all present worms while generates no false positives. There is a trade-off between content block length and the number of generated signatures, and between content block length and the specificity of signatures. (Section 3.4.1)

- **Polymorphic Worm Signatures.** Autograph is not successful in finding signatures if the invariant portion in the worm payloads is relatively short compared to the configured content block size as shown in Section 3.4.2 The extreme case is for a polymorphic or metamorphic worm. We expect that polymorphic worm signatures are not easily expressed with a single payload pattern string even when generated by a human; more expressive forms of signatures [122, 93, 94, 147] are necessary to detect polymorphic worms.

- **P2P Application and Worm Traffic.** They are similar in their communication patterns, so it is hard to distinguish one from another if we rely only on content-prevalence analysis as shown in Section 3.5. Content-prevalence is a good measure to select worm traffic out of other traffic, but combination with other worm detection heuristics is recommended to avoid generating false positives from P2P application traffic.

- **Distributed Port Scanner Detection.** Tattler distributes a large number of port scanner IP addresses without requiring large bandwidth. It also speeds Autograph's suspicious flow selection, so does the signature generation. (Section 4.3 and 4.4)

- **Online Signature Generation.** Our trace-driven simulation results in Section 4.5 show that there is a trade-off between early signature generation and generated signature quality. Combined with tattler and an aggressive port-scanner detection heuristic, the distributed version of Autograph could generate the specific signature of a random scanning worm before it infects 2% of vulnerable hosts.

- **Worm Flow Accumulation Speed.** The size and location of a monitored network is a critical factor that determines the worm flow accumulation speed, so does the signature generation speed in pattern-extraction-based signature generation systems. (Section 5.1) Particularly, for fast detection of localized scanning worms, a monitor must be located close to the vulnerable host population.

- **Distributed Payload Sharing Strategies.** Distributed monitors that share suspicious flows and generated signatures overcome the limitation from the network size and location of a single monitor. We expected the more monitors is the better for fast signature detection but, in the random scanning worm detection experiments (Section 6.1.2), we did not see much speed-up by including more monitors. However, with many monitors, we can accept the detected signature with more confidence even with a low local threshold applied.

  This is because random scanning worms probe all networks evenly. On the other hand, the localized scanning worm detection speed improved as more monitors participate. Considering the computational overhead that the privacy-preserving payload sharing introduces in the system (Section 8.5), it is desirable to use a handful number of monitors for GSD, and use as many monitors as possible for LSD.

- **Prevalent Innocuous Traffic** Our measurements in Section 6.3 showed that some payload pattern strings from legitimate traffic appear common across many networks, but can be filtered from signature generation with a signature blacklist. Moreover, we did find none of these common payload pattern strings include private information.

- HOTITEM-ID **Protocol.** The HOTITEM-ID supports bandwidth efficient, privacy-preserving payload sharing when thousand of monitors participate. From our experiments in Section 7.4, we confirmed its bandwidth efficiency and accuracy. The experiments show another interesting way to deploy a Autograph system. By placing the suspicious TCP flow reassembly part close to the end hosts but enabling the collaborative content analysis on the distributed suspicious flow pool, we can avoid the problem of TCP flow reassembly discussed by Handley *et al.* [57].

- **Privacy-preserving Threshold Set Union.** The original protocol is expensive (Section 8.5). The computational costs increases quadratically to the number of monitors and the number of payload patterns to be shared. Thus, the number of payload pattern strings reported from a monitor must be kept small. Sampling and the extended protocol (Section 8.3.2) helped us make our system practical.

In this dissertation, we focused on Internet worms. However, there are many types of attacks that share the properties of worms. E-mail spams and Blog spams are also wide-spreading and often contain invariant parts in their payload. Thus, we expect our techniques of content prevalence testing and privacy-preserving distributed monitoring might be also useful against these wide-spreading malicious payload attacks.

Appendix A

Extended Protocol for Reducing Network Delay

In Section 8.3, we described a protocol that identifies content blocks that are reported by more than λ_G distributed nodes. The key of the protocol is to represent the set of content blocks each node holds as an encrypted polynomial, multiply all the polinomials from participating nodes in *Collection Phase*, and then find λ_G -th derivative of the resulting polynomial. The collection phase of the protocol in Figure 8.1 presents a way to compute the product of all polynomials by visiting every node sequentially. This is problematic when many nodes (i.e. monitors) participate in the protocol and the average network delay between nodes is high. The delay due to the communication in the collection phase is $O(M \times d)$ where d is the average delay between neighbors and M is the number of participating nodes.

We now discuss an improvement of THRESHOLD-SET-UNION-HBC in order to reduce the communication cost to $O(c \cdot d \cdot log(M))$. Note that this extension requires more homomorphic multiplication and decryption operations, so increases the computational cost in practice. However, this extension might be helpful in different types of applications.

We reduce the communication delay by modifying the collection phase. Multiplication can be done concurrently along a binary tree as shown in Figure A (a), instead of passing computed polynomials sequentially.

For convenience, we assume that the number of participating monitors, M, is a power of two. M_i ($0 \leq i < M$) refers to i-th monitor in the system and f_i is the encrypted polynomial that represents the set of content blocks M_i has observed.

New Collection Phase: i-th monitor M_i where $i \mod 2 \neq 0$ initiates the collection phase by sending the encryption of its polynomial, f_i, to M_{i-1}. For example, M_1 sends its en-

crypted polynomial to M_0.

Upon receiving the polynomial from the neighboring monitor M_{i+1}, M_i ($i \mod 2 = 0$) computes $E_{pk}(f_i \times f_{i+1}) = f_i *_h E_{pk}(f_{i+1})$. Then, if $i \mod 2^2 \neq 0$, M_i sends the encrypted polynomial to M_{i-2^1} as shown in Figure A.

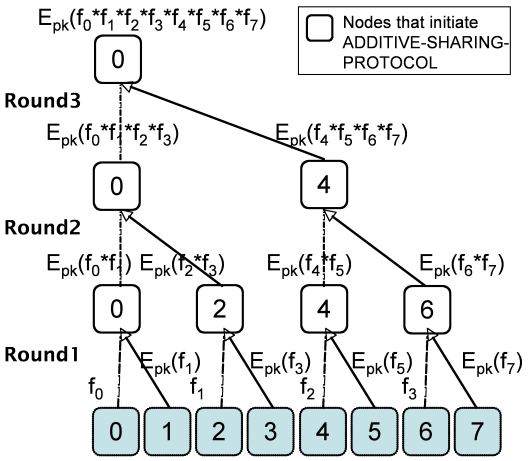
Now node_i ($i \mod 2^2 = 0$) has two encrypted polynomials, $E_{pk}(f_{left})$ and $E_{pk}(f_{right})$ from its left and right subtrees. Because the multiplication of two encrypted polynomials is not possible, M_i first sends a polynomial, $E_{pk}(f_{left})$ to a member of a group of $c+1$ arbitrary nodes. [1] Those $c+1$ monitors compute additive shares of f_{left} using ADDITIVE-SHARE-PROTOCOL presented in Figure A (b).

Let s_i be the share of the i-th node in the group. Then, $\sum_{i=0}^{c} s_i = f_{left}$. M_i sends $E_{pk}(f_{right})$ to the node who holds s_0. The $c+1$ nodes can collaboratively compute $f_{left} *_h E_{pk}(f_{right}) = s_0 *_h E_{pk}(f_{right}) +_h \cdots + hs_{c+1} *_h E_{pk}(f_{right})$. The resulting polynomial is sent to the requesting node. Then, M_i sends the newly computed polynomial to M_{i-2^2} if $i \mod 2^3 \neq 0$. We keep sharing out one subtree's encrypted polynomial and multiplying another subtree's encrypted polynomial with the additive shares along the tree. After $\log M$ rounds, M_0 calculates $E_{pk}(\prod_{i=0}^{N-1} f_i)$ and becomes ready to proceed for the collaborative reduction phase described in the basic protocol.

No coalition of at most c PPT honest-but-curious adversaries can decrypt the polynomial shared with ADDITIVE-SHARE-PROTOCOL because at least $c+1$ nodes are needed to decrypt messages.

This new extended protocol offers the same level of privacy protection as the basic protocol. However, the communication cost reduction comes at the cost of more computations. When $f(x) = f_1(x) \cdot f_2(x)$, and d_1 and d_2 are the degrees of f_1 and f_2 respectively, the number of multiplication operations to compute $f(x)$ is $O(d_1 * d_2)$. The $d_1 * d_2$ is maximized when $d_1 = d2$. In this protocol. the total number of homomorphic multiplication operations is larger than in the original ring-based protocol. Moreover, ADDITIVE-SHARE-PROTOCOL requires each coefficient to be decrypted, so is expensive. Thus, this protocol is useful only when the size of each S_i is small and the number of adversaries c is small.

[1] Note that the M_i itself could be a member of the group. The members of a group could be chosen based on the computational capacity, current load, proximity, etc.



(a) Multiplication tree

Protocol: ADDITIVE-SHARE-PROTOCOL
Input: There are $c+1$ nodes $(n_0, \cdots n_c)$. One of the nodes (say n_0 for convenience, receives two encrypted polynomials, $E_{pk}(f)$ and $E_{pk}(g)$, for multiplication.
Output: $E_{pk}(f*g)$.

- n_i chooses a secret random polynomial r_i of the same degree as f.
- All nodes compute $E_{pk}(f + r_0 + \cdots + r_c)$ and decrypt.
- n_0 combines partial decryptions and obtain $f + r_0 + \cdots + r_c$.
- The additive share s_i of n_i is $-r_i$ if $i \neq 0$. the additive share s_0 of n_0 is $(f + r_0 + \cdots + r_c) - r_0$. Thus, $\sum s_i = f$.
- All nodes compute $E_{pk}((s_i * g)) = s_i * E_{pk}(g)$ and n_0 sums them. The result is $E_{pk}(f*g) = f *_h E_{pk}(g) = (\sum s_i) *_h E_{pk}(g) = \sum s_i *_h E_{pk}(g)$.

(b) Protocol for additive sharing

Figure A.1: **Components of new collection phase in the extended protocol.**

Bibliography

[1] K. Aguilera and R. Strom. Efficient atomic broadcast using deterministic merge. In *Proceedings of the 19th ACM PODC*, 2000.

[2] Akismet. Stop comment spam and trackback spam. `http://akismet.com`, 2005.

[3] Mark Allman, Ethan Blanton, Vern Paxson, and Scott Shenker. Fighting coordinated attackers with cross-organizational information sharing. In *5th Workshop on Hot Topics in Networks (HotNets–V)*, November 2006.

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

[5] S. Antonatos, K. G. Anagnostakis, E. P. Markatos, and M. Polychronakis. Performance analysis of content matching intrusion detection systems. In *Proceedings of the International Symposium on Applications and the Internet (SAINT'04)*, Tokyo, Japan, January 2004.

[6] Ziv Bar-Yossef, T. S. Jayram, Ravi Kumar, D. Sivakumar, and Luca Trevisan. Counting distinct elements in a data stream. In *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*, pages 1–10, London, UK, 2002. Springer-Verlag.

[7] Bittorrent. `http://bitconjurer.org/BitTorrent/`.

[8] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[9] Blubster. `http://www.blubster.com`.

[10] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In *Proceedings of the 24th international conference on cryptology (CRYPTO)*, volume 3152, pages 41–55. Springer-Verlag, August 2004.

[11] Dan Boneh and Hovav Shacham. Group signatures with verifier-local revocation. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 168–177, 2004.

[12] Flavio Bonomi, Michael Mitzenmacher, Rina Panigraphy, Sushil Singh, and George Varghese. Beyond bloom filters: From approximate membership checks to approximate state machines. In *Proceedings of ACM SIGCOMM Conference*, September 2006.

[13] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[14] Andrei Broder and Michael Mitzenmacher. Using multiple hash functions to improve IP lookups. In *Proceedings of IEEE INFOCOM 2001*, pages 1454–1463, 2001.

[15] David Brumeley, James Newsome, Dawn Song, Hao Wang, and Somesh Jha. Towards automatic generation of vulnerability-based signatures. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2006.

[16] M. Cai, K. Hwang, Y.-K. Kwok, S. Song, and Y. Chen. Collaborative internet worm containment. In *Proceedings of IEEE Symposium on Security and Privacy*, May/June 2005.

[17] M. Cai, R. Zhou, K. Hwang, C. Papadopoulos, and S. Song. Wormshield: Collaborative worm signature detection using distributed aggregation trees. Poster on the 2nd Symposium on Network System Design and Implementation (NSDI'05), May 2005.

[18] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. Scribe: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE Journal on Selected Areas in Communication (JSAC)*, 20(8), 2002.

[19] Miguel Castro, Michael B. Jones, Anne-Marie Kermarrec, Antony Rowstron, Marvin Theimer, Helen Wang, and Alec Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proceedings of the Conference on Computer Communications (INFOCOM'03)*, 2003.

[20] CERT. CERT Advisory CA-2004-26 Nimda Worm. `http://www.cert.org/advisories/CA-2001-26.html`, September 18, 2001.

[21] P. Chandra, Raj Yavatkar, and S. Lakshmanmurthy. *Intel IXP2400 Network Processor*, volume 1, chapter 13, pages 259–276. Morgan Kaufmann, October 2002.

[22] D. Chaum. The dining cryptographers problem: unconditional sender and recipient untraceability. *J. Cryptol.*, 1(1):65–75, 1988.

[23] D. Chaum, J.-H. Evertse, J. Graaf, and R. Peralta. Demonstrating possession of a discrete logarithm without revealing it. In *Advances in Cryptology—CRYPTO '86*, pages 200– 212. Springer-Verlag, 1987.

[24] David L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–90, 1981.

[25] Mihai Christodorescu and Somesh Jha. Static Analysis of Executables to Detect Malicious Patterns. In *Proceedings of the 12th USENIX Security Symposium*, 2003.

[26] Yang-Hua Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, pages 1–12, Santa Clara, CA, June 2000. ACM.

[27] B. Chun, J. Lee, and H. Weatherspoon. Netbait: a distributed worm detection service, 2003.

[28] Cisco Systems. Network-Based Application Recognition. `http://www.cisco.com/en/US/products/ps6616/products_ios_protocol_group_home.html`.

[29] Clip2. The Gnutella Protocol Specification v.0.4, March 2001.

[30] C. Jason Coit, Stuart Staniford, and Joseph McAlerney. Towards faster string matching for intrusion detection or exceeding the speed of snort. In *2nd DARPA Information Survivability Conference and Exposition (DISCEX II'01)*, June 2001.

[31] M. Patrick Collins and Michael K. Reiter. Hit-List Worm Detection and Bot Identification in Large Networks Using Protocol Graphs. In *Proceedings of the 10th International Symposium on Recent Advances in Intrusion Detection*, August 2007.

[32] Intel Corp. Intel's tera-scale research prepares for tens, hundreds of cores. Technology-Intel Magazine, `http://www.intel.com/technology/magazine/computing/tera-scale-0606.htm`, 2006.

[33] Symantec Corporation. Symantec security response - w32.sasser.worm. http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html, February 24, 2005.

[34] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, and Lintao Zhang. Vigilante: End-to-end containment of internet worms. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, October 2005.

[35] Jedidiah R. Crandall, Zhendong Su, S. Felix Wu, and Frederic T. Chong. On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 235–248, New York, NY, USA, 2005. ACM.

[36] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local Worm Detection Using Honeypots. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 39–58, October 2004.

[37] P. J. Denning, editor. *Computers Under Attack: Intruders, Worms, and Viruses*. ACM Press, Addison-Wesley, New York, 1990.

[38] Sarang Dharmapurikar, Praveen Krishnamurthy, Todd Sproull, and John Lockwood. Deep packet inspection using parallel bloom filters. In *IEEE Micro*, pages 44–51, 2003.

[39] Direct connect (file sharing). http://en.wikipedia.org/wiki/Direct_Connect_(file_sharing).

[40] John B. Douceur. The Sybil Attack. In *Proceedings of the IPTPS*, Boston, March 2002.

[41] DShield.org. DShield - Distributed Intrusion Detection System. http://dshield.org.

[42] edonkey2000. http://www.edonkey2000.com.

[43] Dan Ellis, Jack Aiken, Kira Attwood, and Scott Tenaglia. A Behavioral Approach to Worm Detection. In *Proceedings of ACM Workshop on Rapid Malcode*, Fairfax, VA USA, October 2004.

[44] Endace. http://www.endace.com, 2004.

[45] Cristian Estan and George Varghese. New directions in traffic measurement and accounting: Focusing on the elephants, ignoring the mice. *ACM Transactions on Computer Systems*, August 2003.

[46] Diego Zamboni Eugene H. Spafford. Intrusion detection using autonomous agents.

[47] Amos Fiat and Adi Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Proceedings on Advances in cryptology—CRYPTO '86*, pages 186–194, London, UK, 1987. Springer-Verlag.

[48] Fileguri. `http://fileguri.com`.

[49] Mike Fisk and George Varghese. Fast content-based packet handling for intrusion detection. Technical report, University of California at San Diego, La Jolla, CA, USA, 2001.

[50] P. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting of lotteries. In *Proceedings of Financial Cryptography*, 2000.

[51] Pierre-Alain Fouque and David Pointcheval. Threshold cryptosystems secure against chosen-ciphertext attacks. In *Proc. of Asiacrypt 2001*, pages 573–84, 2000.

[52] M. J. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *Advances in Cryptography: Proceedings of Eurocrypt*, 2004.

[53] Sharon Gaudin. Storm worm botnet more powerful than top supercomputers. `http://www.informationweek.com/news/internet/showArticle.jhtml?articleID=201804528`, September 2007.

[54] Goboogy. `http://goboogy.com`.

[55] Oded Goldreich. *The Foundations of Cryptography - Volume 2*. Cambridge University Press, May 2004. `http://www.wisdom.weizmann.ac.il/~oded/foc-vol2.html`.

[56] Zoltan Gyongyi, Pavel Berkhin, Hector Garcia-Molina, and Jan Pedersen. Link spam detection based on mass estimation. In *Proceedings of the 32nd International Conference on Very Large Data Bases*, Seoul, Korea, September 2006.

[57] Mark Handley, Christian Kreibich, and Vern Paxson. Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics. In *Proceedings of USENIX Security Symposium*, 2001.

[58] Honeynet Project. Know Your Enemy: Honeynets. `http://project.honeynet.org/papers/honeynet/`, May 2006.

[59] Hotline communications. `http://en.wikipedia.org/wiki/Hotline_Connect`.

[60] Yang hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proceedings of the ACM SIGCOMM Conference*, San Diego, CA, USA, 2001.

[61] Qiang Huang, Helen Wang, and Nikita Borisov. Privacy-preserving friends troubleshooting network. In *Proceedings of the Symposium on Network and Distributed Systems Security*, February 2005.

[62] imesh. `http://www.imesh.com`.

[63] P. Judge and M. Ammar. Gothic: A group access control architecture for secure multicast and anycast. In *Proceedings of IEEE INFOCOM'02*, 2002.

[64] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2004.

[65] Srikanth Kandula, Sankalp Singh, and Dheeraj Sanghi. Argus – A Distributed Network Intrusion Detection System. In *Proceedings of USENIX SANE*, 2002.

[66] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and Kc claffy. Transport layer identification of p2p traffic. In *Proceedings of Internet Measurement Workshop (IMC'04)*, Oct 2004.

[67] Vijay Karamcheti, Davi Geiger, Zvi Kedem, and S. Muthukrishnan. Detecting malicious network traffic using inverse distributions of packet contents. In *Proceedings of ACM SIGCOMM MineNet Workshop (MineNet'05)*, 2005.

[68] Kazaa. `http://kazaa.com`.

[69] Lea Kissner, Hyang-Ah Kim, Dawn Song, Oren Dobzinski, and Anat Talmy. Efficient, secure, and privacy-preserving hot item identification and publication. Technical Report CMU-CS-05-159, Carnegie Mellon University, August 2005. `http://www.cs.cmu.edu/~leak/set-tech.pdf`.

[70] Lea Kissner and Dawn Song. Privacy-preserving set operations. In *Advances in Cryptology - CRYPTO 2005*, Lecture Notes in Computer Science. Springer Verlag, August 2005.

[71] Eddie Kohler, Jinyang Li, Vern Paxson, and Scott Shenker. Observed structure of addresses in ip traffic. In *Proceedings of the SIGCOMM Internet Measurement Workshop (IMW)*, 2002.

[72] Jack Koziol. *Intrusion Detection With Snort*. Sams Publishing, 2003.

[73] C. Kreibich and J. Crowcroft. Honeycomb—Creating Intrusion Detection Signatures Using Honeypots. In *Proceedings of the 2nd Workshop on Hot Topics in Networks (HotNets-II)*, 2003.

[74] Christian Kreibich and Robin Sommer. Policy-controlled event management for distributed intrusion detection. In *4th International Workshop on Distributed Event-Based Systems*, 2005.

[75] Robert Lemos. Counting the Cost of Slammer. CNET news.com. `http://news.com.com/2100-1001-982955.html`, 2003.

[76] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and Brian Chzvez. Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In *Proceedings of IEEE Symposium on Security and Privacy*, May 2006.

[77] Z. Liang and R. Sekar. Fast and automated generation of attack signatures: A basis for building self-protecting servers. In *Proceedings of the 12th ACM Conference on Computer and Communications Security*, 2005.

[78] Patrick Lincoln, Phillip Porras, and Vitaly Shmatikov. Privacy-preserving sharing and correlation of security alerts. In *Proceedings of the 13th USENIX Security Symposium*, pages 239–254, August 2004.

[79] LinkSleeve. LinkSleeve: SLV : Spam Link Verification. `http://www.linksleeve.org`, 2006.

[80] Madster. http://en.wikipedia.org/wiki/Madster.

[81] D. Mazìeres. A toolkit for user-level file systems. In *Proceedings of the USENIX Annual Technical Conference*, Boston, MA, June 2001.

[82] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, October 1996.

[83] Messaging Anti-Abuse Working Group (MAAWG). Email Metrics Program: The Network Operators' Perspective. http://www.maawg.org/email_metrics_report, June 2006.

[84] Alan Mislove, Gaurav Oberoi, Ansley Post, Charles Reis, Peter Druschel, and Dan S. Wallach. AP3: Cooperative, decentralized anonymous communication. In *11th ACM SIGOPS European Workshop*, September 2004.

[85] D. Moore and C. Shannon. Code-Red: A Case Study on the Spread and Victims of an Internet Worm. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop (IMW'02)*, 2002.

[86] David Moore. Network telescope. Technical Report tr-2004-04, CAIDA, 2002.

[87] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the Slammer worm. *IEEE Security and Privacy*, 1(4):33–39, July 2003.

[88] David Moore and Colleen Shannon. The spread of the witty worm. *IEEE Security and Privacy*, 2(4), July/August 2004.

[89] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proceedings of IEEE INFOCOM*, 2003.

[90] Mp2p technologies. http://www.mp2p.net.

[91] Athicha Muthitacharoen, Benjie Chen, and David Mazières. A Low-bandwidth Network File System. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, 2001.

[92] Napster.

[93] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatic signature generation for polymorphic worms. In *Proceedings of IEEE Security and Privacy Symposium*, Oakland, CA USA, May 2005.

[94] James Newsome, Brad Karp, and Dawn Song. Thwarting signature learning by training maliciously. In *Proceedings of the 9th International Symposium On Recent Advances In Intrusion Detection (RAID'06)*, September 2006.

[95] James Newsome and Dawn Song. Dynamic taint analysis: Automatic detection, analysis, and signature generation of exploit attacks on commodity software. In *Proceedings of Network and Distributed System Security Symposium (NDSS'05)*, February 2005.

[96] D. Nojiri, J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition*, April 2003.

[97] Nowcom. Clubbox. `http://www.clubbox.co.kr`.

[98] Nucleus Research, Inc. Spam: The Silent ROI Killer, nucleus research note d59. `http://nucleusresearch.com/research/d59.pdf`, 2003.

[99] P2pradio. `http://p2p-radio.sourceforge.net/`.

[100] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proc. of Asiacrypt 2000*, pages 573–84, 2000.

[101] Ruoming Pang, Vinod Yegneswaran, Paul Barford, Vern Paxson, and Larry Petterson. Characteristics of internet background radiation. In *Proceedings of ACM Internet Measurement Conference (IMC'04)*, October 2004.

[102] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24), 1999.

[103] Sophos Plc. The growing scale of the threat problem, A Sophos white paper. `http://www.sophos.com/security/whitepapers`, February 2006.

[104] Phillip A. Porras and Peter G. Neumann. EMERALD: event monitoring enabling responses to anomalous live disturbances. In *1997 National Information Systems Security Conference*, oct 1997.

[105] Niels Provos. A Virtual Honeypot Framework. In *Proceedings of the 13th USENIX Security Symposium*, August 2004.

[106] T. H. Ptacek and T. N. Newsham. Insertion, evasion and denial of service: Eluding network intrusion detection. Secure Networks, Inc., `http://www.aciri.org/vern/Ptacek-Newsham-Evasion-98.ps`, January 1988.

[107] Ramneek Puri. Bots & Botnet: An Overview. `http://www.sans.org/rr/whitepapers/malicious/1299.php`, December 2003.

[108] Michael O. Rabin. Fingerprinting by Random Polynomials. Technical Report TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[109] S. Rafaeli and D. Hutchison. A survey of key management for secure group communication. *Journal of the ACM Computing Surveys*, 35(3), 2003.

[110] Anirudh Ramachandran and Nick Feamster. Understanding the network-level behavior of spammers. In *Proceedings of the ACM SIGCOMM Conference*, September 2006.

[111] M G. Reed, P. F. Syverson, and D. M. Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communication, Special Issue on Copyright and Privacy Protection*, 1998.

[112] Michael K. Reiter and Aviel D. Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, June 1998.

[113] J. Reynolds. RFC 1135 - Helminthiasis of the Internet, December 1989.

[114] Stuart E. Schechter and Michael D. Smith. Access for sale: A new class of worm. In *Proceedings of the 1st Workshop on Rapid Malcode (WORM'03)*, Washington, DC, USA, October 2003.

[115] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RFC 1889 - RTP: A Transport Protocol for Real-Time Applications, 1996.

[116] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[117] Shareshare. `http://www.shareshare.com`.

[118] John Shoch and Jon Hupp. The "Worm" Programs - Early Experience with a Distributed Computation. *Communications of the ACM*, 25(3):172–180, March 1982.

[119] Sumeet Singh, Cristian Estan, George Varghese, and Stegan Savage. Automated worm fingerprinting. In *Proceedings of 6th Symposium on Operating Systems Design and Implementation*, San Francisco, CA USA, December 2004.

[120] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur. DIDS (Distributed Intrusion Detection System) – Motivation, Architecture, and An Early Prototype. In *14th National Computer Security Conference*, pages 167–176, October 1991.

[121] Robin Sommer and Vern Paxon. Exploiting independent state for network intrusion detection. In *Proceedings of ACSAC*, 2005.

[122] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, October 2003.

[123] Dawn Song. Practical forward secure group signature schemes. In *Proceedings of the 8th ACM conference on Computer and Communication Security*, November 2001.

[124] Soribada. `http://www.soribada.com`.

[125] Lance Spitzner. Honeypots - definitions and value of honeypots. `http://www.tracking-hackerts.com/papers/honeypots.html`, May 2003.

[126] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical Automated Detection of Stealthy Portscans. *Journal of Computer Security*, 10(1-2), 2002.

[127] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, 2002.

[128] Stuart Staniford. Containment of scanning worms in enterprise networks. Journal of Computer Security, 2005.

[129] S. Staniford-Chen, S. Cheung, R. Crawford, M. dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. Grids - a graph-based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference*, October 1996.

[130] Tatter & Company. EAS - Eolin Antispam Service. http://antispam.eolin.com, 2006.

[131] The Snort Project. Snort, The Open-Source Network Intrusion Detection System. http://www.snort.org/.

[132] Robert Vamosi. Botnets for sale. http://reviews.cnet.com/4520-3513_7-6719515-1.html?part=rss&subj=edfeat&tag=Botnets+for+sale, March 2007.

[133] Vipul Ved Prakash. Vipul's Razor. http://razor.sourceforge.net, 1999.

[134] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, 2005.

[135] vshare. http://www.vshare.com.

[136] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. Shield: vulnerability-driven network filters for preventing known vulnerability exploits. In *Proceedings of ACM SIGCOMM Conference*, Portland, OR, USA, August 2004.

[137] Helen J. Wang, Yih-Chun Hu, Chun Yuan, Zheng Zhang, and Yi-Min Wang. Friends troubleshooting network: Towards privacy-preserving, automatic troubleshooting. In *Proceedings of IPTPS*, February 2004.

[138] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *Proceedings of the 7th International Symposium on Recent Advance in Intrusion Detection*, September 2004.

[139] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the 8th International Symposium on Recent Advance in Intrusion Detection*, September 2005.

[140] N. C. Weaver. Warhol Worms: The Potential for Very Fast Internet Plagues. http://www.cs.berkeley.edu/~nweaver/warhol.html.

[141] Nicholas Weaver, Stuart Staniford, and Vern Paxson. Very fast containment of scanning worms. In *Proceedings of the 13th Usenix Security Symposium (Security 2004)*, August 2004.

[142] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, December 2002. USENIX Association.

[143] Davey Winder. Battle of the botnets. `http://www.daniweb.com/blogs/entry1464.html`, May 2007.

[144] Winmx. `http://www.winmx.com`.

[145] J. Wu, S. Vangala, L. Gao, and K. Kwiat. An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'04)*, 2004.

[146] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global Intrusion Detection in the DOMINO Overlay System. In *Proceedings of Network and Distributed System Security Symposium (NDSS 2004)*, 2004.

[147] Vinod Yegneswaran, Jonathon T. Griffin, Paul Barford, and Somesh Jha. An architecture for generating semantics-aware signatures. In *Proceedings of USENIX Security Symposium (Security'06)*, 2006.

[148] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for internet worms. In *Proceedings of the 10th ACM conference on Computer and Communication Security*, pages 190–199, 2003.