A Scalable Hybrid Network Monitoring Architecture for Measuring, Characterizing, and Tracking Internet Threat Dynamics

by

Michael Donald Bailey

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2006

Doctoral Committee:

Professor Farnam Jahanian, Chair
Assistant Professor Zhuoqing Morley Mao
Associate Professor Brian Noble
Professor Gary M. Olson

ABSTRACT

A Scalable Hybrid Network Monitoring Architecture for Measuring, Characterizing, and Tracking Internet Threat Dynamics

by

Michael Donald Bailey

Chair: Farnam Jahanian

Networks are increasingly subjected to threats that affect the reliability of critical infrastructure including Distributed Denial of Service attacks, scanning worms, and botnets. These threats pose significant challenges to measurement infrastructure due to their global scope, extreme scale, and dynamic behavior. As a result, current techniques do not provide sufficiently early or comprehensive intelligence about these attacks. In order to address the problem of providing timely, detailed forensic information on new Internet threats we propose a hybrid system that combines the benefits of network-based and host-based sensors without the corresponding drawbacks. We present insights into the various techniques employed in such a system. We examine the utility of using traffic to unused address space as a means for scalable monitoring for the emergence of new threats and show that while scalable, care must be taken as different sensors see different views of the same global event. We show how the key to achieving scalability is the use of intelligent filtering, allowing the distributed network sensors to selectively send threats to be evaluated to the host sensors based on either the emergence of new threat payloads of the increase in the number of attackers. We highlight

the two major issues in monitoring threats with host sensors; how to configure them, and how to analyze the data. We dismiss the idea that monolithic configurations are sufficient configurations and show how anomaly detection can provide an effective means of automating forensics. Finally we show the impact of combining these two types of sensors is profound, providing an unprecedented level of visibility into Internet threats. We demonstrate this utility by providing examples of both individual threat analysis, and insights into threats such as their escalated threat, increasingly global scope, and persistent population.

To my wife Heather

ACKNOWLEDGEMENTS

This thesis represents a long personal journey spanning not only my 11 years in Ann Arbor, Michigan (you wake up one morning and realize you no longer go to school here, you live here), but also those many years before. It was those before years, I think, which made most susceptible to this kind of self-abuse (I was normal before I met you, I swear!). Since the list of people who have walked with me, guided me, and generally kicked me in the pants to get moving is quite long, I think I had best start at the beginning.

I'd like to thank my mom and dad for giving me life and their unfailing love. I'd like to thank them for the great gifts of my brothers; Robert, Jonathan, and Joshua. You made growing up fun. Your constant reminders about me behaving like the "keeper of all knowledge and wisdom" have served to remind me of the importance of humility. I'd like to thank my *other* brothers Jeff and Joe who taught me that learning can also be fun. I also want to thank all the other members of our constantly growing family including Patti, Andy, Martin, Charlotte, Carl, and Janessa.

I'd like to thank those that came before me in school for breaking in Farnam and leaving a strong legacy of good science; especially Scott Dawson, Craig Labovitz, Robert Malan, and Scott Johnson. It wasn't enough that you all went to school together; you had to work together as well. Thank you for dragging me into it, I had a blast. A big part of that was the great people at Arbor and I would love to a big shout out to all the p33ps on #hackers. In particular, I'd like to thank those at Arbor who supported my personal journey with advice or technical expertise including Danny McPherson, Jose Nazario, Abha Ahuja, and Doug Orr. I am thankful for the hard work of Jenn Burchill that made this a much more readable thesis. I am especially grateful for the previous work on wide address monitoring performed by Dug Song, Robert Stone, and Robert Malan, without which there would be no IMS.

I'd also like to thank the most recent cadre of students, coworkers, and friends who made graduate school so much fun including; Manish Karrir, Andrew Myrick, Sushant Sinha, Niels Provos, Karl Rosaen, David Watson and Evan Cooke. David allowed me to ply him with sage advice when

it might better be applied to myself and “let” me distract him whenever I needed a break. I want to thank Evan for all the ongoing arguments about science, marketing, and life. For two people who hardly ever agree on anything, we have managed to get a lot done.

I wish to thank my advisor and friend, Farnam Jahanian. Without his patience, guidance, and support, I wouldn’t be here. (I’m not sure if I am thanking you or blaming you). Its been one amazing ride and I am honored to have shared part of it with you. I also want to thank the rest of my doctoral committee; Morley Mao, Brian Noble, and Gary Olson for their valuable feedback and encouragement.

Finally, and most importantly, I want to thank Montrez, Lawrence, and Heather. Montrez and Lawrence, it is a blessing you were born and I am happy that none of us have to do it alone. Thank you both for your patience the last few months. Heather there are never enough words to say how deeply I love you. It has meant a great deal to me that you are willing to give up so much to be on this journey with me. There is no one I would rather walk the paths of life with. Are you ready? These are the words you have been longing to hear! I’m done now. Really.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure

CHAPTER 1

INTRODUCTION

Networks are increasingly subjected to threats that affect the reliability of critical infrastructure. These include Distributed Denial of Service attacks, such as the SCO DDoS attacks [14], and scanning worms, such as CodeRed [103] and Blaster [17]. These threats bring with them a new combination of characteristics that make them difficult to track and characterize. First and foremost, they are globally scoped, respecting no geographic or topological boundaries. In addition, these threats can be exceptionally virulent and can propagate to the entire population of susceptible hosts in a matter of minutes. Their virulence is extremely resource taxing and creates side effects that pose problems even for those that are outside the vulnerable population. To make matters worse, these threats have the potential to be zero-day threats, exploiting vulnerabilities for which no signature or patch has been developed. Finally, these threats are evolutionary, changing quickly with the motives and means of the attackers. As a result of these properties, the real world impact of these threats is profound, causing disruptions of real world infrastructure [91] and costing individual institutions hundreds of thousands of dollars to clean up [45].

To address the concerns raised by these threats, researchers have proposed a variety of threat detection and classification systems to provide detailed forensic information on new threats in a timely manner. As many threats propagate by scanning the IPv4 address space, researchers have turned to monitoring many addresses at the same time in order to quickly detect these threats [103, 101]. By monitoring large numbers of addresses, these systems can notably increase the probability of quickly detecting a new threat as it attempts to infect other hosts on the Internet [77]. However, as threats become increasingly complex, interacting with the infected hosts to elicit the important threat features, such as exploit, root-kits, or behavior, may require increasingly complex host emulation. This, coupled with the possibility of zero-day threats that may provide little or no warning

for creating these emulated behaviors, may leave wide address monitoring systems unable to identify the important threat characteristics. In contrast, honeypot systems provide detailed insight into new threats by monitoring behavior in a controlled environment [27, 106]. By deploying honeypot systems with monitoring software, one can automatically generate detailed forensic profiles of malicious behavior [58]. Unfortunately, this detailed analysis comes at the expense of scalability, and hence time to detection. As a result, current techniques do not provide sufficiently early or comprehensive intelligence about these attacks. To be effective at assuring the availability, confidentiality, and integrity of Internet resources, it is necessary to combine information from disparate network and host approaches, each with differing levels of abstraction, into one unified view of a threat.

In order to address the problem of providing timely, detailed forensic information on new Internet threats such as denial of service attacks, worms, and botnets, we propose a hybrid system that combines the benefits of network-based and host-based sensors without the corresponding drawbacks. We contend that the key to achieving this balance is the use of intelligent filtering, allowing the distributed network sensors to selectively send threats to be evaluated to the host sensors. The impact of combining these two types of sensors is profound, providing an unprecedented level of visibility into Internet threats and serving as the basis for a wide variety of practical applications including threat detection, signature generation, threat mitigation or quarantine, and threat clean-up.

1.1 Approach

In this thesis we present our research into measuring, tracking, and characterizing Internet threats. The first component of this thesis is the design of a hybrid architecture that combines techniques from both network-based and host-based threat measurement into a unique system capable of providing useful information about Internet threats. In describing the hybrid architecture we discuss the tradeoffs between various exiting measurement techniques and show how these systems typically trade off detailed information about a threat (what we call depth) with speed of detection and threat coverage (what we call breadth). We show that a carefully designed system that makes use of both network and host techniques can be effective at achieving both depth and breadth without the corresponding costs. We also enumerate the list of open research problems that need to be solved in creating such a system.

The second component of this thesis is the design, deployment, and evaluation of an innovative network monitoring component for the hybrid system. Based on the novel idea of watching traffic

to unused or unreachable address space, we have created a system that is capable of observing many of the artifacts of Internet threat behavior. Expanding upon existing work in this field, we created a distributed sensor network that consists of numerous unused blocks in geographically and topologically diverse locations. In order to recover additional information about Internet threats, this system makes use of a new stateless TCP responder to illicit the first payload packet from propagating threats. In describing this novel sensor network, show one of the key problems associated with these sensors in the context of a hybrid system; visibility. We show the surprising result that different sensors see different things, even when what they are observing are supposedly global events. Therefore, to be successful, a sensor network must have a degree of diversity.

The next element of this thesis is the intelligent filtering component. In order to achieve the benefits of both host-based and network-based monitoring of threats without the corresponding drawbacks, the hybrid system must be intelligent about managing its resources. In our approach, connection attempts seen at the network-based sensors are filtered before being sent to the host-based sensors for analysis. Two novel techniques are discussed, based on the ideas of both content and source prevalence. These techniques for reducing the volume of data seen at the sensor blocks are a fundamental contribution for hybrid systems, as it reduces not only the volume of data to be analyzed by the host sensor system, but also the offered connection load.

The fourth component of this thesis is the design, deployment, and evaluation of an innovative host monitoring component for the hybrid system. The host component of the hybrid system is responsible for providing detailed forensic information regarding the various threats observed by the networking component. We describe a host monitoring system based on a centralized collection of honeypots, often called a honeyfarm. We describe our architecture, which consists of components for managing virtual machines, as well as three novel threat detectors that look for host behavior across a variety of host resources. We address two of the fundamental questions regarding the honeyfarm operation. First, we discuss the problem of determining what services to run on the honeyfarm, and in doing so dismiss the idea that a simple monolithic honeyfarm is sufficient. We show that to be effective, the honeyfarm should create representative and consistent profiles of the target enterprise, or global host population. Second, we examine the problem of characterizing threats based on the host data. Since the honeyfarm can generate a large number of events, some amount of filtering is required. We provide several filtering techniques and show when these heuristics techniques fail.

The final component of this thesis is a characterization of various Internet threats using the tech-

niques presented as part of the hybrid system. Using the hybrid system and its various components we have been able to provide a wealth of information about Internet threats, including worms, denial of service attacks, and botnets. While each of these analyses provide important security and operational knowledge, when taken in aggregate, they provide a picture of the continuing Internet threat evolution. Therefore, we also provide some of these insights, including trends in threat demographics, payloads, and persistence.

1.2 Background

To better familiarize readers with the concepts underlying our research, this section presents an overview of Internet threats. We present the key terms and concepts that introduce the foundation for our work.

1.2.1 Internet Threats

As national utility infrastructures become intertwined with emerging global data networks, their stability and the integrity of the two have become synonymous. This connection, while necessary, leaves network assets vulnerable to the rapidly moving threats of today's Internet, such as denial of service attacks (DoS), Internet worms, and Botnets.

1.2.1.1 Denial of Service Attacks

Coordinated denial of service attacks are considered one of the most significant threats jeopardizing the explosive growth of the Web. The growing reliance of businesses and consumers on the Web for accessing information and for conducting transactions has made continuous availability of enterprise networks and web sites extremely critical.

The Internet provides a means for connecting a global set of distributed users with network services. Many of these services are *open services*—services that are accessible by anyone. A company's e-commerce Web server is a common example of an open service. Other examples include a company's incoming email, netnews, domain name service, FTP servers, etc. These open services are accessible by anyone with a network address—one of the cornerstones of the Internet model. However, as open services they are equally open to abuse. Coordinated DoS attacks overwhelm these open services with illegitimate traffic, denying access to real users.

The widely publicized denial of service attacks in February 2000 brought down numerous major

Web sites including Yahoo, ZDNet, eBay, Amazon.com, CNN.com, and E-Trade. These coordinated attacks flooded the Web sites with mock traffic from computers around the globe, essentially blocking access to legitimate requests from customers for several hours. Less publicized attacks happen everyday around the clock, and there is no end in sight. A recent report predicts that security breaches in the form of corporate espionage, cyberactivism, and cyberterrorism will become more prevalent. Approximately 2,000 Web sites offer free attack tools, and every day three new software exploits are discovered. It is expected that denial of service attacks will become increasingly common.

Denial of service attacks generally involve organizations whose computers and networks have been compromised. The attacker first scans millions of computers on the Internet to identify unsecured hosts to be used as "launch pads." He then secretly installs software on a master computer and a collection of compromised "zombie" computers. The attacker hides their true identity and location by using these zombie machines to launch the attack. He only needs to have the master signal the zombies to begin the attack, specifying the attack's target and type. The result of this malicious activity is the denial of service to legitimate users because the coordinated attacks often:

- Overwhelm the target Web server with illegitimate requests, thereby choking off the sites available bandwidth,
- Clog the victim's network infrastructure (routers, proxy servers, etc.), or
- Crash the victim's Web server.

1.2.1.2 Internet Worms

Since their initial appearance in the late 80s, network worms have been considered a threat to the availability and security of network computers and the Internet. While the definition of a worm has changed over time, perhaps one of the most succinct and telling definitions of a worm comes from Spafford in his seminal analysis, as he describes the difference between a virus and a worm [105]:

> "A worm is a program that can run by itself and can propagate a fully working version of itself to other machines. It is derived from the word tapeworm, a parasitic organism that lives inside a host and saps its resources to maintain itself. A virus is a piece of code that adds itself to other programs, including operating systems. It cannot run independently; it requires that its "host" program be run to activate it. As such, it has

a clear analog to biological viruses those viruses are not considered alive in the usual sense; instead, they invade host cells and corrupt them, causing them to produce new viruses."

A worm then is a self-propagating piece of malware that exploits a vulnerability in some operating system or application software. In [123] the authors provide a taxonomy by which various Internet worms can be differentiated. They include characteristics based on:

- *target discovery*. This is the process whereby the worm selects the next victim it tries to infect. For example, the worm may randomly scan IP space or select a target from the host's ARP cache.
- *carrier*. Carrier describes how the exploit chooses to propagate. For example, it may choose to actively infect other machines, or to follow only existing communication patterns.
- *activation*. This is the way in which the worm code actually starts execution. For example, immediate execution, tied to a human behavior, or scheduled for a specific time.
- *payloads and attackers*. Payloads and attackers are intimately tied. Payload represents what, aside from propagation, the worm contains. For example a worm may delete files, or open up the ability for a remote attacker to control the infected machine. These payloads reflect the will of the worm designer and the motivations of those who employ them. For example, some common motivations include fame, financial gain, or destruction.

1.2.1.3 Botnets

Global Internet threats are undergoing a profound transformation from attacks designed solely to disable infrastructure to those that also target people and organizations. This alarming new class of attacks directly impacts the day-to-day lives of millions of people and endangers businesses and governments around the world. For example, computer users are assailed with spyware that snoops on confidential information, spam that floods email inboxes, and phishing scams that steal identities.

At the center of many of these attacks is a large pool of compromised computers located in homes, schools, businesses, and governments around the world. Attackers use these *zombies* as anonymous proxies to hide their real identities and amplify their attacks. *Bot* software enables an operator to remotely control each system and group them together to form what is commonly referred to as a *zombie army* or *botnet* [60]. The scope of the botnet problem is difficult to quantify,

as the highly covert nature of bots and botnets makes them difficult to identify and even harder to measure. Nevertheless, CERT has described botnets with more than 100,000 members, and almost one million bot infected hosts have been reported [50, 73]. Operators of business and government networks are beginning to take notice [10, 34].

1.2.2 Internet Threat Dynamics

Unfortunately, the properties of these threats make them particularly difficult to address.

- First and foremost, they are **globally scoped**, respecting no geographic or topological boundaries. For example, at its peak, the Nimda worm created five billion infection attempts per day, which included significant numbers from Korea, China, Germany, Taiwan, and the US [103].

- In addition, they can be exceptionally **virulent** and can propagate to the entire population of susceptible hosts in a matter of minutes. This was the case during the Slammer worm, in which the majority of the vulnerable population (75K+ susceptible hosts) was infected in less than 30 minutes [76]. This virulence is extremely taxing on network resources and creates side effects that pose problems even for those that are outside of the vulnerable population, such as the routing instability associated with the Slammer worm [63].

- To make matters worse, these threats have the potential to be **zero-day** threats, exploiting vulnerabilities for which no signature or patch is available. For example, victims of the Witty worm were compromised via their firewall software the day after a vulnerability in that software was publicized [100].

- Finally, these threats are **evolutionary**, changing in both means and motives as the threat landscape and incentive structures evolve. For example, consider the profound transformation that is currently underway from attacks designed to disrupt to attacks that take control. A recent survey indicated that there were over 900,000 infected bots being used by attackers and that phishing attacks are growing at 28% per month [10].

1.3 Related Work

With so many threats to global Internet security, characterizing, monitoring, and tracking these threats is quickly becoming critical to ensure of individual organizations and the Internet as a whole

run smoothly. Traditional approaches to threat monitoring fall into two broad categories: host-based monitoring and network-based monitoring.

1.3.1 Host Monitoring

Host monitoring is primarily defined by the fact the instrumentation, sensing, and data collection occur at individual hosts. Host-based techniques fall into two basic approaches: forensics and host-based honeypots. Anti-virus software [28] and host-based intrusion detection systems [44] seek to alert users of malicious code execution on the target machine by watching for patterns of behavior or signatures of known attacks.

Host-based honeypots [27, 29, 87, 112] track threats by providing an exploitable resource and monitoring it for abnormal behavior. A honeypot is a non-productive resource that can be monitored to track the activity of attackers and identify other anomalous processes. A major goal of honeypots [106] is to provide insight into the motivation and techniques behind these threats. While these host-based monitoring techniques can provide detailed information about threats, they are limited in their ability to scale to monitor large address blocks. Host-based honeypot systems have traditionally been allocated a single IP address which limits visibility into processes such as random scanning threats [106].

1.3.2 Network Monitoring

In contrast to host monitoring, network monitoring is primarily defined by the fact the instrumentation, sensing, and data collection occur in the network and cover the collection of multiple individual hosts.

1.3.2.1 Active Network Monitoring

One network monitoring approach uses active network perturbation to determine the scope and propagation of threats. This is typically done to elicit a behavior that is only visible by participating in a network or application session. Historic approaches to the active detection and characterization of network-based security threats fall into two categories: monitoring production networks with live hosts and monitoring unused address space (or darknets).

1.3.2.1.1 Active monitoring of used and reachable addresses space Active monitoring of used and reachable addresses takes many forms. There are many research groups involved in the deploy-

ment of Internet probe machines for the measurement of Internet paths and topologies, including the NIMI [67], Surveyor [8], and IPMA [53] projects. These projects have approached Internet measurement by utilizing active performance metrics [54]—measurements that perturb the network—such as one-way loss along a datagram stream, or periodic `traceroutes` between the probe and specific end hosts. Other techniques have monitored routing protocol behavior by participating in the routing protocols themselves [62]. Finally, beacons have been used for the ongoing study of BGP dynamics [69] by both participating in the routing protocols and by actively probing the network.

1.3.2.1.2 Active monitoring of unused or unreachable addresses space Monitoring large numbers of unused addresses simultaneously provides quicker and more complete information about threats than monitoring a single host [32, 61, 77]. However, gathering the same detailed forensic information produced by a real honeypot is a scalability challenge. One approach is to trade fidelity for scalability by emulating operating systems and services rather than running real operating system or application instances [86, 126]. Projects like The Internet Motion Sensor [14] and iSink [126], and software like honeyd [86] are used to bring up networks of emulated honeypots while still balancing the scalability of dark address monitors while gathering more detailed information about threats.

1.3.2.2 Passive Network Monitoring

In contrast to active techniques, passive network monitoring approaches attempt not to intrude on the existing operation of the network. Passive techniques can measure live networks or unused or unreachable address space.

1.3.2.2.1 Passive monitoring of used and reachable addresses space By far, the most common technique is the passive monitoring of production networks. These techniques fall into three main categories: monitoring data from security or policy enforcement devices, monitoring data from traffic characterization mechanisms, and monitoring the traffic directly through sensing or sniffing. By either watching firewall logs, looking for policy violations, or by aggregating IDS alerts across multiple enterprises [117, 97, 125], one can infer information regarding a worm's spread. Other policy enforcement mechanisms, such as router ACLs, provide course-grained information about blocked packets. Instead of dropping these packets, CenterTrack [111] leveraged the existing routing infrastructure to collect denial of service traffic for analysis. Data collection techniques from traffic

planning tools offer another rich area of pre-existing network instrumentation useful in characterizing threats. Course-grained interface counters and more fine-grained flow analysis tools, such as NetFlow [51], offer another readily available source of information. In monitoring used networks, systems may choose to watch traffic directly [68].

While monitoring live networks provides broader coverage, these techniques often face difficulties in distinguishing between benign and malicious traffic.

1.3.2.2.2 Passive monitoring of unused or unreachable addresses space In contrast to monitoring live hosts, darknet monitoring consists of deploying sensors that monitor blocks of unused (or dark) address space. Because there are no legitimate hosts in a darknet, any observed traffic destined to such darknet must be the result of misconfiguration, backscatter from spoofed source addresses, or scanning from worms and other network probing. Recall that methods for watching individual addresses with sacrificial hosts are often called honeypots [27, 106]. Techniques for monitoring much wider address blocks have a variety of names including network telescopes [77, 75], blackholes [104, 103], and darknets [41]. It should be noted that a limitation of this approach is that it relies on observing the target selection behavior of threats. As a result, threats that do not scan for new hosts, or threats that are specifically tailored to avoid unused blocks are not observed. Nevertheless, these techniques have been used with great success in observing denial of service activity [78], worms and scanning [101, 14, 17, 76], as well as other malicious behavior [82].

In contrast to host-based techniques, passively monitoring unused address space scales to very large address spaces. However, this scalability often comes at the expense of not gathering details about specific events.

1.3.3 Mapping Host and Network Monitoring to Threat Characterization

The techniques described above provide varying amounts of intelligence regarding a threat. Some systems capture all the events involved in a particular incident while, others record only a connection attempt. Some systems only have visibility into local events, while others are capable of monitoring globally scoped threats. These tradeoffs, which we refer to as depth and breadth, are bound by their associated costs.

Breadth refers to the ability of the system to detect threats across host, and across operational and geographic boundaries. At one extreme of the breadth axis is the threat view of a single host while at the other is the view of all network-based threats on a global scale. One early approach

to achieving globally scoped visibility was the measurement of wide address blocks [75, 126, 104]. This technique is attractive in that it can easily view a large percentage of the total IPv4 address space and has been effective at characterizing new threats [76, 100]. However, given the finite size of the IPv4 address space, it is important to explore new methods of obtaining breadth. The IMS addresses the issue of breadth by creating a distributed architecture consisting of numerous topologically diverse sensors.

Depth defines the extent to which a sensor emulates the services and characteristic of a real host, similar to the interaction spectrum in honeypots [106]. At one extreme of the depth axis is an instrumented live host, while at the other is the completely passive capture of packets. Multiple points in this spectrum can be utilized simultaneously, as shown by Barford *et al.* [126]. The IMS, however, uses an extension to passive techniques [75] that gains additional intelligence without emulating services to the extent of previous active methods [107, 85].

In isolation, these techniques fail to completely address the scalability and behavioral fidelity requirements needed to monitor these threats. The scope of existing host-based techniques, such as host-based honeypots, anti-virus software, and host-based intrusion detection, is too small to capture global information such as the size of the infected population, or provide warning early in the growth phases. On the other hand, globally-scoped network sensors, such as network telescopes, do not interact sufficiently with the worm. As such, they lack enough information to characterize the vulnerability exploited and its effect on local machines. To be effective at assuring the availability of Internet resources, it is necessary to combine information from disparate network resources, each with differing levels of abstraction, into one unified view of a threat.

1.3.4 Hybrid Monitoring

Acknowledging this need for both host and network views, two new approaches of combining these resources have evolved, portal systems that aggregate fine-grained sensor measurements from a large number of sensors, and hybrid systems that use darknet monitors to concentrate connections to a centralized honeyfarm. Portal projects that aggregate data fall into two categories, those based on aggregating firewall logs or Intrusion Detection System (IDS) alerts across multiple enterprises [117, 113, 125], and those based on constructing and aggregating data from large numbers of honeypots [107, 86].

Hybrid systems [56, 126, 98] combine the breadth of network-based monitoring with the depth of host-based monitoring. There are three projects of interest, and they vary in how they perform

the physical connection funneling, where and in what way they choose to filter the data, and in the diversity and amount of address space monitored.

Collapsar is a centralized honeypot mechanism that relies on routing infrastructure to GRE tunnel IP space back to the honeypots [56]. One key differentiator between these approaches is that our hybrid approach focuses on filtering the interactions before they reach the host sensors, providing lighter requirements for the host sensors. In addition, because our hybrid approach responds to requests topologically next to the surrounding address space, it is not as susceptible to latency based fingerprinting as the tunneling approach.

Another relevant project is that of iSink [126]. iSink focuses on scaling high interaction data collection to wide address ranges such as /8 blocks. There are two main areas in which this work differs. First and foremost, we are evaluating a distributed collection infrastructure that monitors numerous distributed blocks of varying size. Second, we use content prevalence [15] and source prevalence [16] as a mechanism for deciding when and how to filter interactions, unlike iSink which discusses only source destination filtering [82].

Of particular relevance is the recent work on the Potemkin Virtual Honeyfarm [119] in which the authors discuss a hybrid architecture with emphasis on a novel set of techniques for creating scalable per-connection virtual machines. Their scalability gains are achieved by multiplexing across idleness in the network and by exploiting redundancies in the per-host state of the virtual machines using *copy-on-write* virtual machines. This work is complimentary to ours in that we focus on limiting the number of connections seen by the honeyfarm, but the Potemkin authors focus primarily on servicing these connections as efficiently as possible.

1.4 Contributions

The main contributions of this thesis are:

- **Design of a hybrid framework and methodology for quickly and comprehensively characterizing Internet threats**. Current techniques do not provide early or comprehensive intelligence about Internet attacks. The scope of existing host-based techniques is too small to capture useful information, such as the size of the infected population, or provide warning early in the growth phase. On the other hand, network sensors do not interact sufficiently with the worm and lack enough information to characterize the vulnerability exploited or the effect on local machines. We argue that a hybrid architecture that combines multiple tiers of

sensors can achieve the benefits of each tier, without the corresponding drawbacks. Our architecture uses low cost, network-based honeypots as filters to reduce the load of higher-cost, high-interaction honeypots. We argue that this architecture is effective at creating detailed and timely intelligence about Internet threats and can serve as the foundation for other useful activities such as new threat detection and signature generation.

- **Construction and deployment of a network monitoring component for monitoring dark address space**. As part of our research, we have built the Internet Motion Sensor (IMS). IMS is a network of distributed blackhole sensors that monitors blocks of unused address space. IMS consists of over 60 darknets in 30 organizations including academic institutions, corporations, and Internet service providers. This deployment represents a tremendous amount of diverse address space including over 17 million routeable addresses with blocks in over 20% of all routed /8 networks. Each blackhole sensor in the distributed infrastructure monitors a dedicated range of unused IP address space and has an active and passive component. The passive component records all packets sent to the sensor's address space. The unique active component responds to TCP SYN packets with a SYN-ACK packet to elicit the first data payload on all TCP streams.

- **Construction and deployment of a host monitoring component for monitoring dark address space**. The Host Motion Sensor (HMS) is designed to provide additional forensic analysis in response to the changing threat landscape. It consists of four components: a virtual machine management module, a network detection module, a process detection module, and a file detection module. Individual host operating system and application configurations are run on viral machines with the detection modules providing quick detection and characterization of new threats. Automated tools quickly determine if a transaction is worthy of additional analysis.

- **Evaluation of the framework**. We have focused our efforts in the evaluation of the framework on answering several key questions: How does distributing sensors effect visibility? How do you scale sensors? How do you configure sensors? How do you characterize sensor data?

  - *Understanding sensor visibility*. We have explored issues associated with inferring global threat properties from a set of distributed blackhole sensors. In particular, we

demonstrated that significant differences exist in observations made at sensors of equal size that are deployed at different locations. These differences were demonstrated to persist despite accounting for specific propagation strategy. To be successful, then, and network-based monitoring system must account for these differences and provide multiple perspectives.

– *Filtering techniques for the automated reduction of traffic at network sensors*. We examined several of the key characteristics of traffic at individual darknets that effect the scalability of a hybrid system. In particular, we showed that darknets are dominated by a small handful of source IP addresses contacting the same destination ports repeatedly. We then showed, however, that this does not hold across darknets; neither the source IP addresses, nor to a lesser extent the destination ports, appear frequently across darknets. While some of this behavior can be explained by existing work on darknet size, scanning rate, and time to detection, we showed that much of it is the result of the targeted nature of the attacks observed as well as the short on-time of many random scanning hosts. This lack of overlap implies that every additional new block added to a distributed darknet monitor will likely bring with it its own sensor-specific events. For a large collection of darknets to be effective, a new view of events and new methods of filtering are required. We then constructed a filtering mechanism that takes these two factors into account by including distributions of source IP addresses, rather than the source IP addresses specifically, and the number of darknets observing an activity. We showed that this algorithm is effective by examining several recent events, such as activity associated with new WINS vulnerability, Veritas vulnerabilities, and the recent MY-SQL worm.

– *Host-based threat classification using OS-level information flow tracking*. The main challenge of tracking information flow at the operating system level for hybrid honeypots systems is the number of events to be evaluated. While monitoring host-level information flow does provide a great deal more information about threat behavior, it also generates a larger amount of information to process. As a result, we have considered heuristic techniques which may jeapordize breaking dependency chains, but offer greater information reductions. We have provided illustrative examples of when these heuristics work and when they fail.

– *Evaluating the need for representative and consistent host configurations*. The mono-

lithic structure of desktop operating systems has yielded the false impression that a collection of a small number of operating system and application configurations can produce a honeyfarm that is able to capture the majority of Internet threats. We show that in fact, any reasonably sized network shows a surprising large number of operating system, application, and configuration parameters. We show that in order to be effective, these honeypots need to be configured to provide individually consistent as well as globally representative service configurations.

- **Application of the framework to the characterization of Internet threats**. Recall the main motivation of this work was to successfully characterize Internet threats. Therefore, perhaps the most significant contribution of this work is its ability to accomplish just that. These characterizations fall into two main categories: the evaluation of individual events, and the collection of those characterizations into a view of threat trends.

  - *Event or threat-specific data analysis*. Over the course of the system's operation, the hybrid architecture has been able to characterize a wide number of individual events. These include, but are not limited to; worms (e.g., CodeRed, Nimda, Slammer, Blaster, Witty, Slammer, Bobax, and MySQL), denial of service attacks (e.g., SCO attacks), scanning (e.g., WINS, Veritas, Bagle-Backdoor), and Botnets.
  - *Insights into threat trends and the motivation for Internet attacks*. The individual characterizations discussed above, examined in aggregate, have allowed us to gain insight into threat trends and motivation behavior. We have been able to quantify the activities of the four phases of the life cycle of worms, examine trends in threat demographics, worms' persistence, and the escalation of threat payloads.

1.5 Structure of Thesis

This thesis is organized into seven main chapters. The next chapter, Chapter 2, presents the architecture of the hybrid system and an overview of the approaches and challenges associated with hybrid architectures. We then examine the network monitoring component of the hybrid architecture in Chapter 3 and discuss the most important result of our distributed deployment: sensor visibility. In the next chapter we then discuss our work in building filtering algorithms to reduce the number of connections or events that must be evaluated at the host-based sensors. In Chapter 5 we discuss

the host component of the hybrid architecture, present our work on classification of host events, and determine which services to run. Next, we present our experiences in operating various components of this hybrid architecture in Chapter 6 and examine our contribution in analysis of individual threats and threat trends. In the final chapter we summarize our conclusions and contributions and provide future directions for this work.

CHAPTER 2

THE HYBRID ARCHITECTURE

To effectively characterize Internet threats a monitoring system must combine detailed forensics with quick detection and broad coverage. Existing monitoring systems, such as those discussed in the previous chapter, fail to meet this goal. Network systems achieve quick detection and broad coverage at the expense of detailed forensics, while host monitoring systems provide detailed forensic information without any timeliness gurentees. To address this shortfall, we propose a hybrid system consisting of network-based, host-based sensors, and intelligent filtering that addresses both of these design goals.

In this chapter, we discuss the design of this hybrid architecture. We begin with a detailed discussion of the design tradeoffs in a hybrid system of this kind. We then present the architecture itself, along with its two main components: the network component and the host component. We then provide several illustrative examples of how the data from such a system can be used to solve existing security problems such as threat detection, forensics, and mitigation. We conclude by enumerating the main research issues associated with building any such system.

2.1 Design Tradeoffs

In this section, we discuss the tradeoffs in design between high-interaction and low-interaction honeypots. Not all types of monitoring systems are capable of providing insight into threats in the same way, even if they see the same traffic. Each makes a set of tradeoffs with respect to the number of hosts that can be monitored in how many locations (breadth) versus the degree to which the system is capable of behaving like a real system (depth). For example, a system that simply records packets is quite easy to deploy and can scale to very large amounts of traffic, but it would

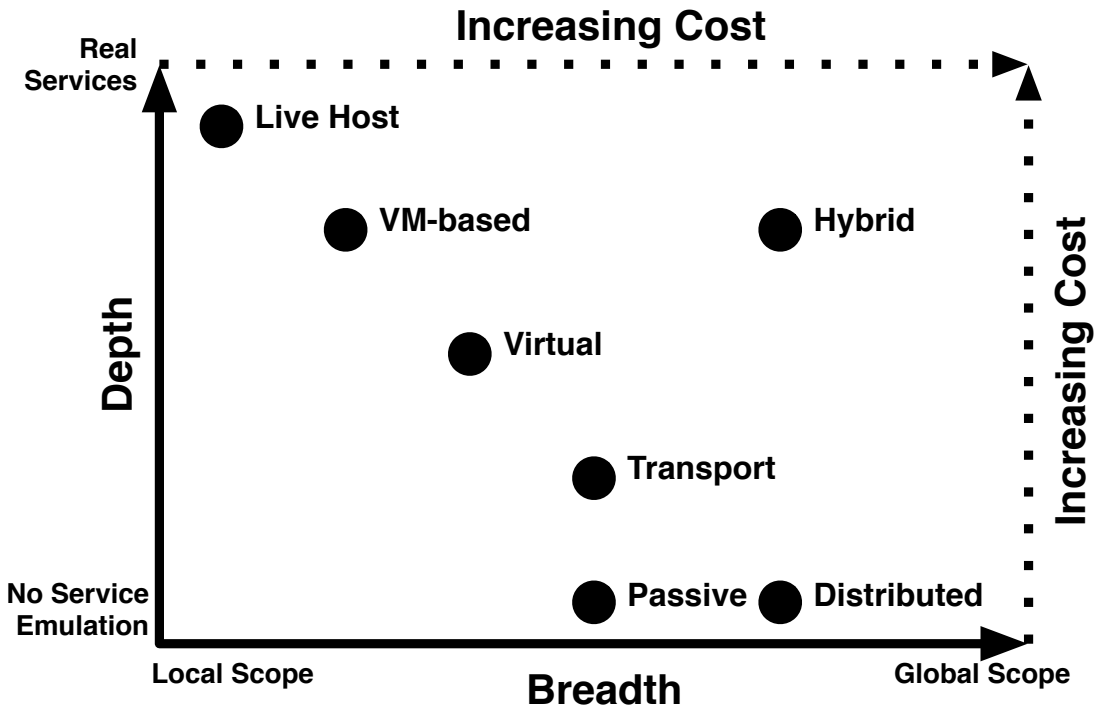


Figure 2.1: The trade off between breadth and depth chosen by various measurement classes.

not be able to differentiate separate threats attacking the same destination port. A live host used as a monitor may easily differentiate threats to the same port as it runs the vulnerable software and is capable of actually being broken into (and monitored for such an event), but administering hundreds of thousands or millions of such machines is difficult.

Figure 2.1 shows how various types of emulation trade off breadth and depth. At the simplest, there are passive DarkNets, which simply record packets to the unused address space. Distributed passive sensors perform the same function, but across multiple organizations. For example, IMS diversified from a single, wide-address DarkNet monitor to a distributed monitor in an effort to expand its coverage of events. Transport layer DarkNets make use of knowledge about transport protocols, such as TCP, to elicit some application information without specific knowledge about what application runs on top of them. For example, the SYN-ACK responder of IMS responds to all TCP SYN requests in an attempt to elicit the first payload packet of a connection. Next, there are network honeypots and service emulators–systems that try to emulate large numbers of operating systems or applications by encoding application and operating system knowledge into the responder. Finally, the highest amount of depth is provided by what are traditionally known as honeypots–hosts running actual operating systems and applications. In an effort to scale these systems, large collections of host operating systems and applications can be run in hardware or software virtual machines, making them easier to deploy and maintain.

2.1.1 Behavioral fidelity (depth)

Behavioral fidelity refers to the degree to which any infrastructure is able to emulate the interactions and behavior of an end host node. Higher fidelity implies a closer approximation of end host behavior. Spitzner et el. employ a similar notion of interactivity to help identify the means and motivations of individual hackers [106, 107]. In the context of a global detection and warning system, behavioral fidelity is important as various degrees of emulation are required to capture and differentiate different types of worms.

Table 2.1 provides an illustrative view of the behavioral fidelity levels and their impact. The first column shows measurement class and the second columns shows the level of interaction at that class. The third column provides an example system that provides that level of fidelity (in general, higher fidelity provides lower levels of interaction as well). The forth column shows a captured threat that required at least that level of interactivity to differentiate the threat from other threats. The final column shows the size of minotored block a single host can scale to using this technique,

Measurement Class	**Minimum Interaction**	**Example Tool**	**Example Threat**	**Scale**
Passive	Capture connection attempt	UCSD Network Telescope[75]	Sapphire[76], Code-Red[101], Witty[100]	/8
Transport	Response on 1 or more ports	IMS[33]	Blaster[80]	/16-/8
Virtual	Application Response	Honeyd[86]	Slapper[11]	/16-/12
VM-Based	Virtualized end host behavior	Potemkin[119], Honeystat[42]	Doomjuice[115], Dabber[114]	/21-/19
Live Host	End host behavior	Honeynet[107]	Agobot[30]	/32

Table 2.1: Honeypot interactivity level required to capture remote exploit.

based on published results. The first row shows threats that can be caught simply by capturing the connection attempt. These threats act on a port that has no legitimate traffic or contain headers that are uniquely identifiable. This may also include UDP and ICMP threats in which the payloads or exploits occur in one packet. The second row shows threats that require a system to respond on a port. These are TCP threats that require session construction to elicit the first message, or threats that OS fingerprint by looking for active ports. The next higher level of interactivity are those threats that require application response. These threats require specific responses due to application semantics or limited scope of vulnerability (e.g., checking for a specific version of Apache before attempting sending an exploit). Finally, we have threats that require full host behavior, such as those threats that attack previous threats or threats for which the created back-doors and behaviors are more important to detect than the threat propagation itself. Note that for scalability reasons, deployments may choose to virtualize these end host services, a technique which can be fingerprinted and avoided.

Regardless of the chosen level of emulation, it is clear that there will always remain some threat that requires a closer approximation of end-host behavior. Therefore, to be comprehensive, a system must include actual hosts as part of its monitoring infrastructure.

2.1.2 Coverage (breadth)

Coverage refers to the ability of an infrastructure to effectively capture globally-scoped events. Moore has shown that increased monitor size yields increased visibility in terms of time to detection and infected population size [77]. Unfortunately, IPv4 space is limited, and there are a small number of wide address blocks available for instrumentation. Therefore, increasing coverage requires combining available smaller sized blocks. While it is certainly the case that increasing the size of monitored space in this way will increase visibility, it may also be the case that true coverage can only be achieved by examining large blocks and collections of disparate blocks. It has been shown that address blocks in different networks see different threat traffic [32]. This effect may be the result of distribution of the affected population, the target selection function, statistical variance, security policy, or the underlying physical topology.

Increased monitoring size decreases the time to detect the global onset of a threat. Different, topologically diverse blocks see different views of these threats. Because of these two factors, we believe that effective coverage is achieved through a distributed collection of sensors that monitor as much address space as possible in as many diverse locations as possible. This notion is often at odds with behavioral fidelity in that scaling high-interaction honeypots to tens of millions of hosts is problematic at best. To be effective, one must balance these two goals.

2.2 Hybrid Architecture

To provide both behavioral fidelity and global, broad coverage, we have proposed a hybrid architecture that is highly scalable but still delivers very accurate detection. We know that lightweight virtual honeypots or darknets can instrument a large address space effectively. On the other hand, we know that they do not support full behavioral fidelity (e.g., a new threat may fail to be accurately captured by a low-interaction honeypot). This is almost the reverse for high-interaction honeypots: a new threat can successfully interact with such a machine, but the high-interaction system does not have optimal performance and would not scale to a large address space. We would like to take advantage of the scalability provided by the low-interaction virtual honeypots, while still being able to provide detailed behavioral analysis. This hybrid architecture consists of three components:

1. Internet Motion Sensor (IMS). Scalable low-interaction honeypots, or darknets, are widely deployed and monitor numerous variable-sized, unused network blocks. When new activity is

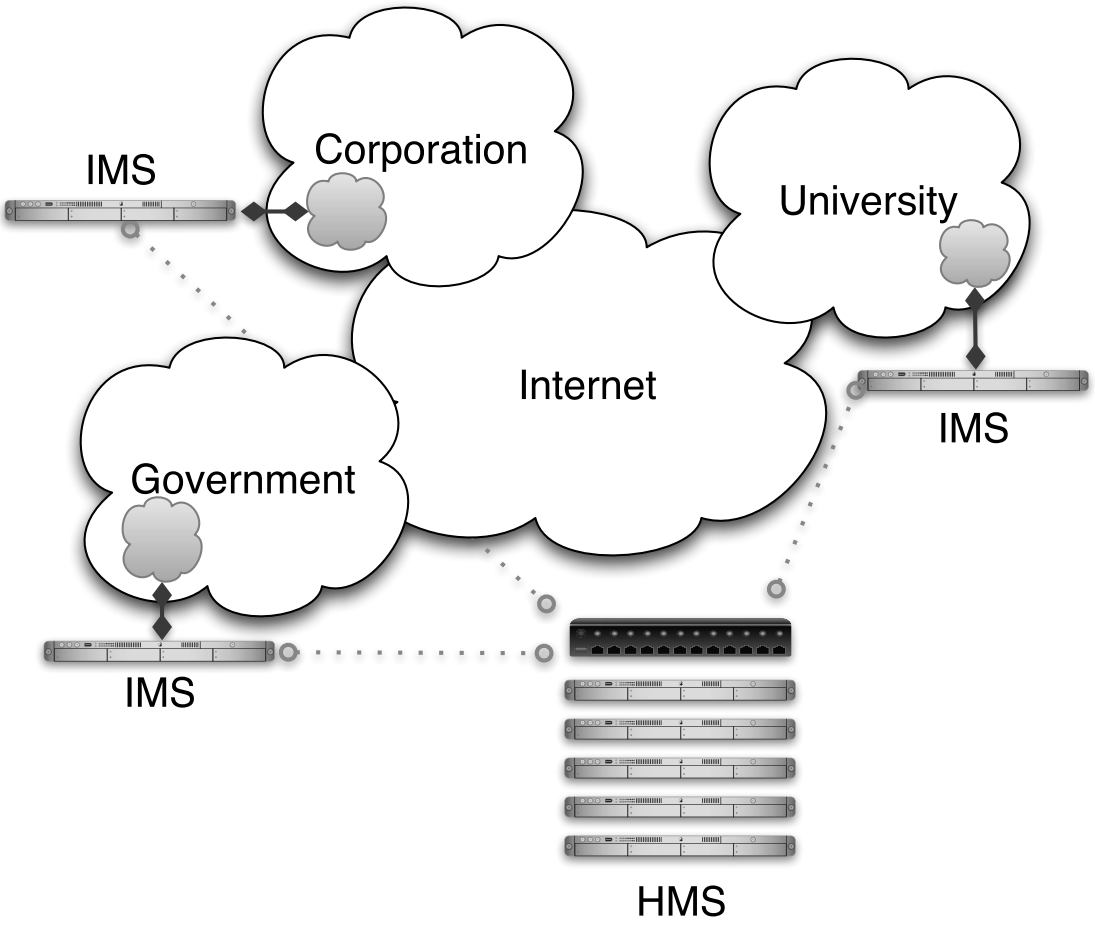


Figure 2.2: A Hybrid architecture with the distributed Internet Motion Sensor (IMS) and the centralized Host Motion Sensor (HMS).

detected by the IMS, the connection is proxied back to the HMS for further in-depth analysis. The connection is relayed to virtual machine images running the application appropriate to the connection request.

2. Host Motion Sensor (HMS). A collection of centralized VMware [120] (virtual machine) hosts that run a wide variety of host operating systems and applications. Filtered traffic from IMS is directed here for behavioral analysis. Thus, new and important threats are handed off and actually executed so the resulting activity can be monitored for new worm behavior.

3. Command and Control. A centralized mechanism provides a unified view of the collection infrastructure including prevalence of sources and payloads. This mechanism coordinates connection redirection across sensors. It also monitors back-end performance and provides feedback to the front-end redirection mechanisms.

2.2.1 Internet Motion Sensor

By watching darknets, the traffic seen by the Internet Motion Sensor is pre-filtered to eliminate both the false positives in identifying malicious traffic and the scaling issues of other monitoring approaches. To analyze this traffic, the IMS sensors have both active and passive components. The active component responds to TCP SYN packets with a SYN-ACK packet to elicit the first data payload on all TCP streams. When a packet is received by a darknet sensor, the passive component computes the hash of the payload. If the hash doesn't match any previously observed signatures, then the payload is stored and the signature is added to the signature database. The Internet Motion Sensor architecture was introduced in 2005 [14], and it has been used to track new threats [17, 13] and to evaluate the importance of distributed darknet monitoring [32].

2.2.2 Host Motion Sensor

The Host Motion Sensor (HMS) is designed to provide additional forensic analysis in response to the changing threat landscape. HMS currently runs VMware to provide several high-interaction systems per physical machine. The HMS network setup allows for connection back to the infecting host, but limits the ability of the honeypots to contact the outside world. The internals of each host consists of two components: a network detection module and a host resource module. The network module, like the outbound connection profiling HoneyStat system [42], is responsible for looking for infected host behavior. It profiles the originating traffic from the honeypot and alerts

on any outbound connection attempt not part of its normal behavior. Publicly available intrusion detection software [95] that matches traffic against known malicious signatures is used as an oracle for classifying any newly detected threats as "known" or "unknown." The final module is the host resource profiler. This system uses BackTracker [58] to build file and process dependency trees to both detect violations of policy and provide detailed forensic information on the files and processes created by an possible infection. Again, existing techniques are used to help identify new activities; in this case, host anti-virus software [28].

2.2.3 Coordination

Effective coordination between components of the hybrid architecture is necessary to achieve the best possible balance between accuracy and scalability. The final piece of the architecture then, is a control component. The control component aggregates traffic statistics from IMS and monitors the load and other data from the HMS. The control component is responsible for analyzing all data received for abnormal behavior, such as a new worm propagating. We achieve this by two separate analysis phases that are combined at the controller. One phase analyzes the network data of the front-ends for statistics that relate to worm propagation (e.g., increases in source IP addresses for a certain port and increases in payload data). The other phase keeps track of the behavior of the back-ends. If we notice that the back-ends initiate unusual network connections or unusual host resource behavior, that is normally a sign of worm infection.

This model also takes care of running target operating systems on top of a virtual machine. This module starts a clean OS image for each infection attempt, and saves the infected state to disk for use by the other modules. Once any malicious activity is detected, the checkpointing features of VMware not only allow us to save these infected states and corresponding forensic information, but they also return the machines to a known good state for further use in analysis. Occasionally, these boxes may need to be reverted back to a compromised state in order to monitor secondary infections. In addition to the specific task of matching future connection attempts with previously installed back doors, this component also manages the working set of applications and operating systems.

2.3 Example Applications of the Hybrid Framework

In this section, we discuss the applicability of this framework to solving some of the standard problems in Internet threat detection and resolution.

2.3.1 Detection

Detection is the process of detecting and alerting network operators and researchers of brewing threats in the network. Traditional network approaches to this problem look at the amount of traffic, source and destination distribution, or the prevalence of content, to determine the advent of a new threat. Host-based techniques rely on anti-virus, intrusion detection, or changes to the underlying operating systems to detect intrusions. Both of these provide value, but they do not provide a high degree of confidence for previously unseen self-propagating code. Network sensors do not support full compromise (e.g., a new worm would fail to propagate by just talking to a low-interaction honeypot). This is almost the reverse for high-interaction honeypots: a new worm can successfully infect such a machine and use it to propagate. Unfortunately, high-interaction systems do not have optimal performance and would not scale to a large address space.

One unique application of the hybrid framework is in the area of worm detection. Recall that the back-end component of this system consists of actual hosts serving as honeypots. Honeypots by their very nature do not have any real users that generate requests or load. The observable actions of the honeypot (network, filesystem, resources, etc.) are the result of either the standard operation of the OS or of external inputs. While honeypots have typically looked at traffic inbound to a honeypot to detect intrusions, worm detection is concerned with self-propagating code–code that exploits a vulnerability and uses the new host as a new infection vector. Instead of watching the inbound traffic for signatures or unsolicited behavior, our novel worm propagation detection mechanism actually watches for the propagation of worms. As mentioned above, honeypots do not open outbound connections routinely as part of their operation and those connections that do occur are easily profiled. Therefore any *outbound* connection is a better indicator of an infection than an inbound analysis. In addition, because worms often use the same propagation method from host to host, we can apply the same content checksumming algorithm to packets out of the back-end honeypot and match them to the MD5 of the inbound connection. A matching outbound signature that matches an inbound handoff signature is even a higher indicator of self-propagating code. Figure 2.3 shows an example of a worm that would be easily caught by this method.

```
GET /default.ida?XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX%u9090%u6858%uc
bd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%
u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP
/1.0..Content-type: text/xml..Content-length: 3379 ........`....
....dg.6..dg.&.......h......\...P.U...\...P.U..@.....X....U.=...
....=.............T....u..~0...........F0.........CodeRedII...$
.U.f.....8.....P.......j...P...P..8...P.E..p.........8....thS.U.
.U..E.i.T...,.....,.............F4.E.Pj..u...........j.j..U.P.U
.Ou..;...i.T....\&....\&.W.U.j.j..U.j..U....F4)E.jd.U...<...P.U.
...<...=....s....>......s.f..p.....f..r....P.d.....t...j.j.j..U.
...t..E.j.Th~f...u..U.Yj...p...P.u..U.........tK3..U.=3'..u?..h..
.......l.........`........E...d.....h...Pj...`...Pj.j..U..j.Th~f
...u..U.Y...u1.....X-....j.h....P.u..U.=....u.j.j...\...P.u..U..
u..U.........w...........xu......`......d$.dg....Xa..dg.6..dg.&
..f.;MZu..K<.<.PE..u..T.x...B..<.KERNu...3.I.r ...A..<.GetPu..J.
I...J$........J.......D$$dg....Xa..Q....]..E......LoadLibraryA..
u..U..E......CreateThread..u..U..E......GetTickCount..u..U..E...
...Sleep..u..U..E......GetSystemDefaultLangID..u..U..E......GetS
ystemDirectoryA..u..U..E......CopyFileA..u..U..E......GlobalFind
AtomA..u..U..E......GlobalAddAtomA
```

Figure 2.3: The payload for CodeRed II.

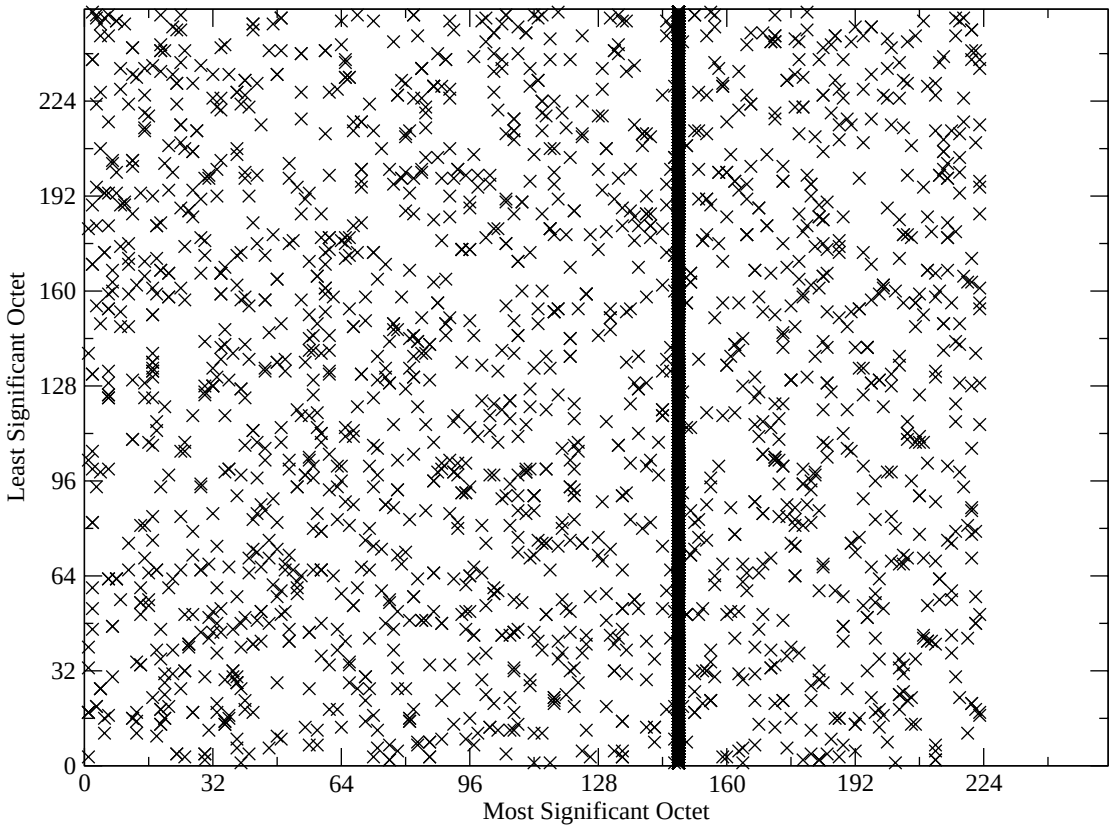


Figure 2.4: The scanning patterns of CodeRed II.

This new approach is desirable because it combines the unique features of each sensor type. Actual infections provide a higher degree of confidence in actual self-propagating code, but they do not scale to large enough monitored address blocks to be useful for early detection. Another interesting artifact of this approach is that there is now an infected host running the worm code. This code can be decompiled, the filesystem of the infected host can be compared to a known good state, and the numerious behavioral and resource components of the worm can be analyzed.

2.3.2 Forensics

Forensics is about learning about previous threats to help understand the motivation and methods used by hackers. The two different world views, as expressed by traditional network and by honeypot communities, have lead to two different approaches to forensics. The networking community characterizes geographic demographics, infected population size, global packet-per-second

rates, etc. These variables are studied in order to determine the impact of worms on specific parts of the networking infrastructure. On the other hand, host-based communities have focused on techniques for detecting intrusions, for performing the intrusions, for detection evasion, and for other host-specific resource behavior. While both sets of information are useful and provide insight into these threats, there are categories of analysis that require both perspectives to be effective.

Consider, for example, building a complete understanding of worm propagation. One approach to this problem would be to look at individual IP address or organizations and the infection attempts they make on various parts of the address space as observed by network sensors. The other approach to the same problem is to watch infection attempts from the host perspective. This can be accomplished either by monitoring network connections from a live host or by capturing worm payloads in the laboratory and studying the propagation code. By themselves, each approach suffers from specific limitations. Network-centered monitoring is statistically unable to understand the global perspective. It's impossible to measure network traffic for every destination address, and even large blocks will be left unmonitored. This means that specific hotspots might be missed by this approach. In contrast, host-based approaches are unable to account for policy filtering devices and/or network bottlenecks and outages that would affect the actual propagation of the worm.

Figure 2.4 shows the worm propagation, from the host perspective, by plotting the first 100,000 connection attempts from a jailed honeypot infected with the CodeRed II worm. The graph shows the relationship between the first and last octet of a destination address for an attempted worm infection. Octet graphs of this nature are designed to show correlations in destination address selection and highlight address selection preferences. In the case of CodeRed II, we see three interesting results. First, CodeRed II never selects /8 address blocks greater than 224. While this behavior is not revealed in code analysis, this class D space may not be routed due to the default routing behavior of a Windows machine. Second, CodeRed II has a strong preference to the local /8 (and also the local /16, although a separate octet graph is needed to show this). In this graph, this manifests itself as a dense collection of points along the vertical line for the /8 block. Finally, a barely visible gap is seen at the 127/8 block, as this /8 is also skipped by CodeRed II.

What this doesn't show is that even when these well-defined propagation strategies are known through host-based analysis, observations at network sensors are wildly deviant from the controlled behaviors. Previous work has shown that these differences and conjectures are the result of errors in target selection, policy, topology, and statistical variance [32]. Without capturing both the observed network behavior and the host behavior, it is difficult to obtain a global perspective on worm

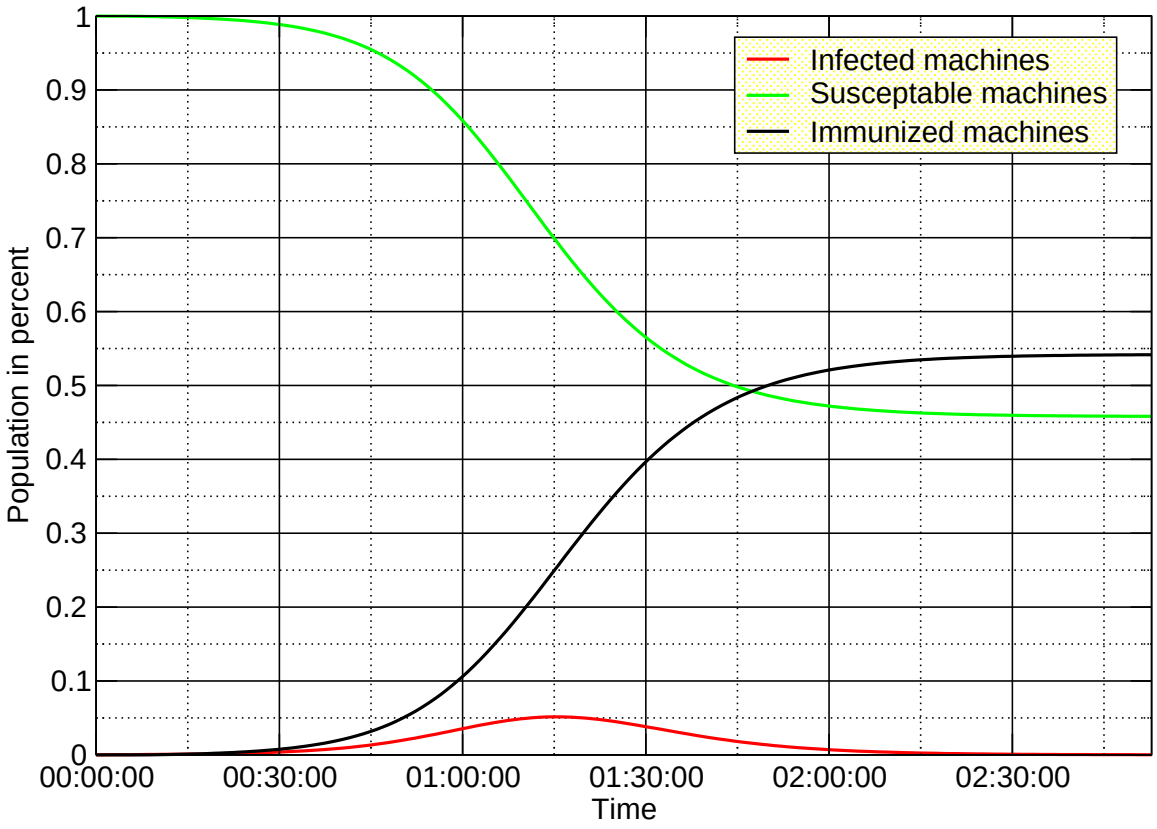


Figure 2.5: A simulated example of worm propagation when employing virtual honeypots to immunize infected hosts.

propagation. This combination of both the network and host sensors enables not only propagation strategy, but also the system's ability to determine the infection mechanism and key indicators, or signatures, that are needed to protect against future infections and clean up existing infections.

2.3.3 Signature generation and mitigation

Signature generation is the process of defining all the necessary characteristics of a new threat to be able to detect a new occurrence of the threat, identify existing infected hosts, and immunize against additional infections. This process is uniquely suited to the hybrid architecture as it requires a sufficient number of monitored hosts to catch the threat early in its growth phase and sufficient detailed behavioral analysis to identify previously unseen threats.

As an example, consider the cause of an auto immunizing system. Analogous to Moore et el., we can model the effect of immunization on worm propagation by using the classic SIR (Susceptible, Infectious, Recovered) epidemic model. The model states that the number of newly infected hosts

increases linearly with the product of infected hosts, the fraction of susceptible hosts, and the contact rate. The immunization is represented by a decrease in new infections that is linear in the number of infected hosts.

Figure 2.5 (courtesy of Niels Provos) shows a simulated example that tracks the change in the susceptible, infected, and immunized populations. In these example we assume 360,000 susceptible machines in a 32-bit address space, an initial worm seed of 150 infected machines, and a worm probe rate of 50 probes per second. The simulation measures the effectiveness of using active immunization by virtual honeypots. The honeypots start working after a time delay. The time delay represents the time that is required to detect the worm and install the immunization code. We expect that immunization code can be prepared before a vulnerability is actively exploited. Time to detection and signature creation are key limited factors in this approach. If we wait for an hour, all vulnerable machines on the Internet will be infected. Our chances are better if we are able to begin immunization after 20 minutes. In that case, a deployment of about 262,000 honeypots is capable of stopping the worm from spreading.

2.4 Issues in Hybrid Systems

While clearly providing promise as a method for quickly and comprehensively characterizing Internet threats, there are several research problems associated with this or any hybrid approach that must be solved in order to ensure successful operation. These include:

- **Filtering Interactions.** Large darknets see staggering amounts of traffic that cannot simply be redirected to a honeyfarm. In order to achieve scale, the amount of data presented to the honeyfarm must be significantly reduced. We have approached the problem of filtering by designing techniques that filter based on the prevalence of packet contents [15], as well as on the number of sources [16].

- **Emulating Real Host Behavior.** As threats become increasingly sophisticated, detection systems must become correspondingly complex to elicit the proper response and to avoid fingerprinting. Understanding this arms race and the associated tradeoffs is key to any successful deployment.

- **Automating Forensics.** At the speed new threats are capable of spreading, the operational impact of human-scaled forensic analysis is minimal. Automated techniques are needed for

generating actionable forensic information about a threat, including behavioral signatures describing activity at the network and/or host levels.

- **Managing Virtual Machines.** Even with advanced filtering mechanisms, the virtual machines are expected to handle a large number of requests and must be managed efficiently (as is discussed in the Potemkin Virtual Honeyfarm [119]). In addition, the effectiveness of the entire system is dependent on its ability to accurately represent the vulnerable population of interest.

2.5 Summary

In this chapter, we have presented our architecture for combining network-based and host-based measurement into a single hybrid system capable of quickly and comprehensively measuring Internet threats. We have provided a framework for discussing the various design tradeoffs and have shown that systems make tradeoffs between the breadth, depth, and cost of measurement. We introduced the design of a hybrid system consisting of the Internet Motion Sensor and the Host Motion Sensor. We argued that this hybrid system is capable of achieving balance between breadth and depth through the use of intelligent filtering. We showed how data from such a hybrid system can have a far-reaching impact on the availability and security of the Internet. We concluded by enumerating the main research issues associated with hybrid systems.

In the following two chapters, we detail our evaluation of the Internet Motion Sensor and Host Motion Sensor and our efforts to address these core research issues.

CHAPTER 3

NETWORK MONITORING IN THE HYBRID ARCHITECTURE

Breadth in the hybrid architecture is achieved through the monitoring of numerous unused network blocks in topologically diverse locations. To achieve this goal, we have designed, deployed, and evaluated the Internet Motion Sensor (IMS) system. Initially deployed in 2001, IMS monitored a single large address block. Today, IMS is a globally distributed monitoring system consisting of over 60 address blocks across three continents. IMS sensors are in over 30 different topological locations in 19 autonomous systems (ASes). These networks cover a wide range of organizations, including tier-1 service providers, regional service providers, broadband networks, academics institutions, and Fortune 500 enterprises. Each IMS sensor monitors a block of unused address space, varying in size from a /25 (128 IP addresses) to a /8 (16 million IP addresses), and the entire system monitors roughly 1% of routed IPv4 space. The IMS project has proven exceptionally useful in Internet security research and has prompted a variety of published results including those on the IMS architecture [14], sensor placement [32], filtering [16], and individual threats [17, 34].

In this chapter, we discuss our experiences with the Internet Motion Sensor as a means for achieving breadth in our hybrid architecture. We begin by introducing the architecture of IMS. Next, we discuss each of its unique components including its distributed architecture, its novel payload caching, and its lightweight active responder. Finally, we examine one of the more interesting results from our distributed deployment–that different network sensors provide different perspectives into global threats.

3.1 Related Work

In order to quickly and accurately characterize the emergence of new threats, researchers have investigated a number of novel network monitoring techniques. One popular approach involves recording packets destined to globally routed but unused address space. Since there are no legitimate hosts in an unused address block, any observed traffic destined to such addresses must be the result of misconfiguration, backscatter from spoofed source addresses, or scanning from worms and other network probing. This technique has a variety of names including network telescopes [77], blackhole monitors [103], DarkNets [41], Sinkholes [47], or background radiation [82]. This technique has been instrumental in capturing important information about diverse threats, such as denial of service attacks [78], random scanning worms [101, 76, 17, 100], and botnets [34]. Existing work involving this technique has focused primarily on architectures [14, 77, 126, 119], characterizations of traffic [82, 32, 16], and analysis of specific threats [78, 101, 76, 17, 100, 34], rather than the operation of these systems.

3.2 Internet Motion Sensor Architecture

The Internet Motion Sensor was designed to measure, characterize, and track a broad range of Internet threats. This particular challenge necessitates a lightweight monitoring approach having global visibility. More specifically, the IMS was designed to:

- Maintain a level of interactivity that can differentiate traffic on the same service.
- Provide visibility into Internet threats beyond address, geographical, and operational boundaries.
- Enable characterization of emerging threats while minimizing incremental effort.

While other systems have combined sensors of different measurement fidelities [126], the goal of our project is to gain global threat visibility, rather than in-depth information on the specific mechanisms of a threat. This implies that a simple system that captures less data may actually be more effective than a complex system producing large amounts of data. In other words, it is possible to trade in-depth threat information for the ability to gain additional visibility.

The tradeoff of visibility over in-depth threat information motivates the architecture described in this section. The IMS consists of a set of distributed blackhole sensors, each monitoring a dedicated

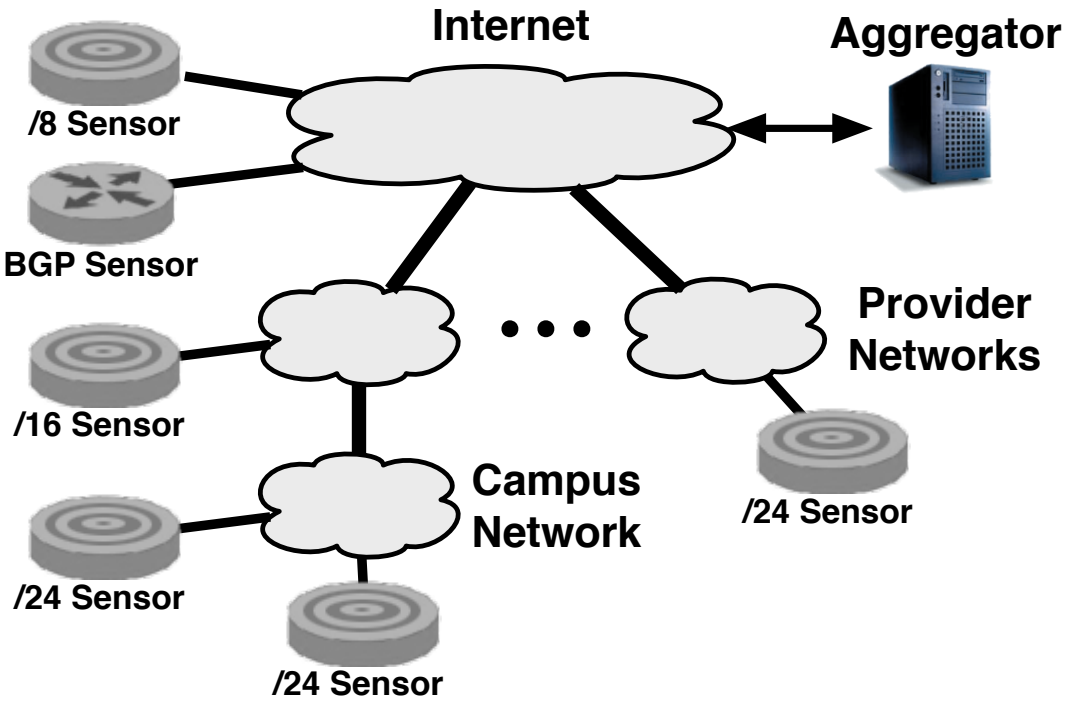


Figure 3.1: The Internet Motion Sensor architecture.

range of unused IP address space. Because there are no legitimate hosts in an unused address block, traffic must be the result of misconfiguration, backscatter from spoofed source addresses, or scanning from worms and other probing. The blackhole sensors in the IMS have an active and a passive component. The passive component records packets sent to the sensor's address space, and the active component responds to specific packets to elicit more data from the source.

The active component is designed to elicit the first payload of data across the major protocols (TCP, UDP, and ICMP). UDP is a connection-less protocol so application-level data is sent without the receiver ever responding. For example, the Witty [100] and Slammer [76] worms were based on UDP in which the entire worm payload was transmitted in the first packet. TCP, on the other hand, is a connection-oriented protocol and requires an active response to elicit any payload data. The IMS uses a simple lightweight active responder to establish a TCP connection and capture payload data on TCP worms like Blaster [80] and Sasser [39]. ICMP data is passively collected.

Storing the full payload for every packet has significant space requirements, so the IMS uses a novel payload storage approach. When a blackhole sensor receives a packet with a payload, it first computes the MD5 checksum of the payload (without network headers) and compares it against the checksum of all the other packets it has seen in the past day. If the checksum (signature) has already been recorded, the capture component logs the signature but does not store the payload. If the signature is new, the payload is stored and the signature is added to the database of signatures seen that day.

This architecture offers three novel contributions:

- **Distributed Monitoring Infrastructure:** The IMS is designed from the ground up for distributed deployment to increase visibility into global threats, including those that may illustrate targeting preferences.

- **Lightweight Active Responder:** The active responder is designed to maintain a level of interactivity that can differentiate traffic on the same service-independent of application semantics. This enables IMS to characterize threats on emergent ports and services without additional deployments or scripting.

- **Payload Signatures and Caching:** The IMS checksums and caches commonly seen packets such that only new payloads need be stored. This saves significant storage resources and enables a simple mechanism for identifying new payloads.

The following three subsections describe and validate these novel components of the architecture by showing them in the context of the distributed IMS deployment consisting of 28 address blocks.

3.3 Distributed Blackhole Network

A key contribution of the IMS is the creation of a large distributed sensor network built from address blocks of various sizes and placed in a variety of topologically diverse locations. While previous attempts at monitoring unused address blocks have focused on three or fewer address blocks [103, 75, 126, 82], our approach is to be widely distributed. This architectural choice has two key advantages: greater visibility into distant events and broader coverage of threats.

Using the analogy of Astrometric Telescopes, Moore in [75] notes that the greater the size of a light telescope, the greater the visibility into fainter, smaller, further and older objects. Moore notes that the more address space monitored (for random scanning events) the better the insight into shorter lived or lower rate events. The conclusion is that having large address blocks is important for monitoring globally scoped events. In addition to wide address blocks, Moore also suggests distributed blocks as a method for increasing visibility.

Distributed sensors provide more addresses that increase visibility and also provide another important benefit, broader coverage of threats. The following investigation is motivated by the observation that there are significant differences in the traffic observed in the distributed IMS blackhole sensors. Because many Internet threats today, like many worms, are globally scoped, one might expect somewhat similar traffic on equally sized blackholes. Recall that unlike live networks, traffic must be the result of misconfiguration, backscatter from spoofed source addresses, or scanning from worms and other types of probing. Despite the global nature of these threats and the lack of legitimate traffic, the IMS distributed sensors see widely different amounts of traffic along several important dimensions.

This section probes these differences by using three successively more specific views of traffic to a network of distributed blackhole sensors. The data was recorded over a one-month period with SYN responders on TCP ports 135, 445, 4444, and 9996 across all sensors. These ports were chosen in order to monitor specific worms. The first view looks at the traffic observed at each sensor over all protocols and services. The second view looks at traffic to a specific protocol and port. Finally, the third view looks at the signature of a known worm over all sensors.

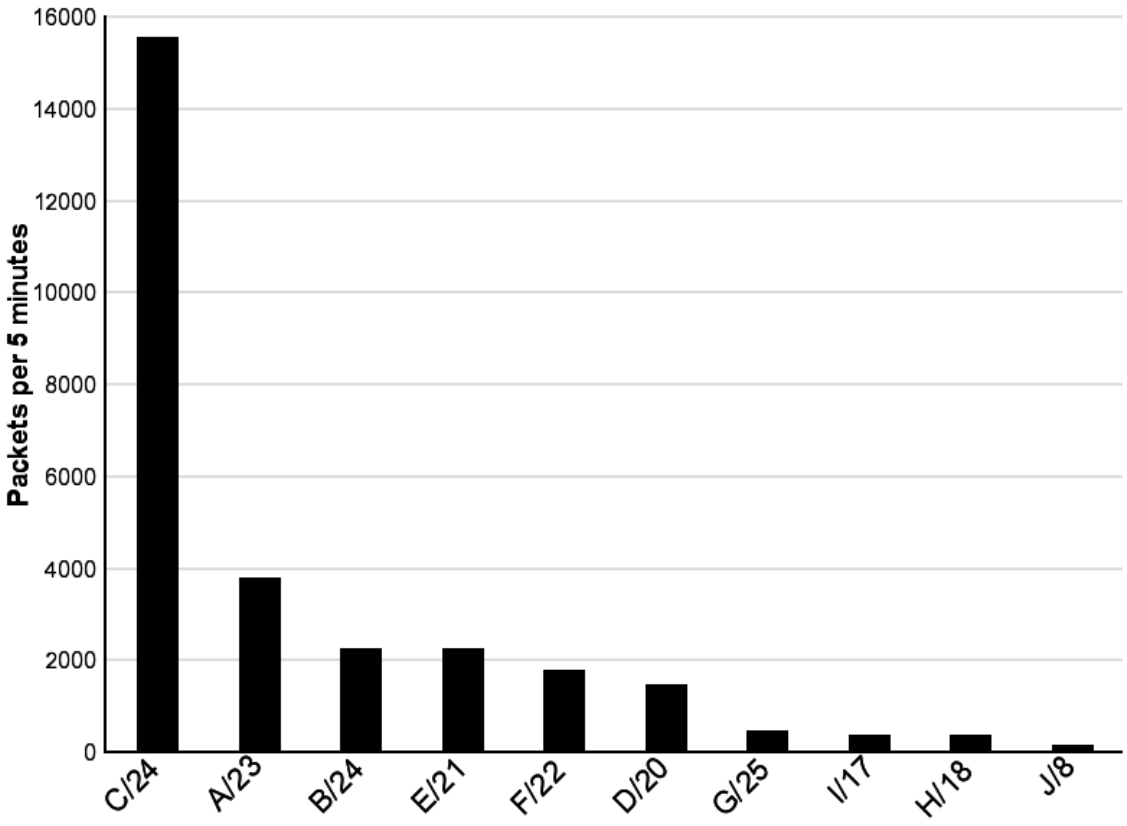


Figure 3.2: Packet rate as seen by each sensor, normalized by /24.

3.3.1 All protocols and services

We begin by looking at the packet rate observed by each sensor. Figure 3.2 shows the average amount of traffic over all protocols and services observed by ten blackhole sensors. Packets are normalized by the size of a /24, so sensors covering different sized blocks can be compared. Normalization is performed such that the magnitude seen in a /23 would be divided by two, and traffic in a /25 multiplied by two. The granularity of a /24 was chosen to focus the discussion on larger differences between blocks rather than individually targeted addresses. While differences likely exist within individual /24 blocks, this is beyond the focus of this thesis.

Figure 3.2 clearly shows that the amount of traffic varies dramatically and can differ by more than two orders of magnitude between sensors. Of note, the larger blocks typically observe less traffic per /24 than the smaller blocks. One possible explanation is that the smaller blocks are closer in the IPv4 address space to live hosts than the large blocks. There are many reasons a

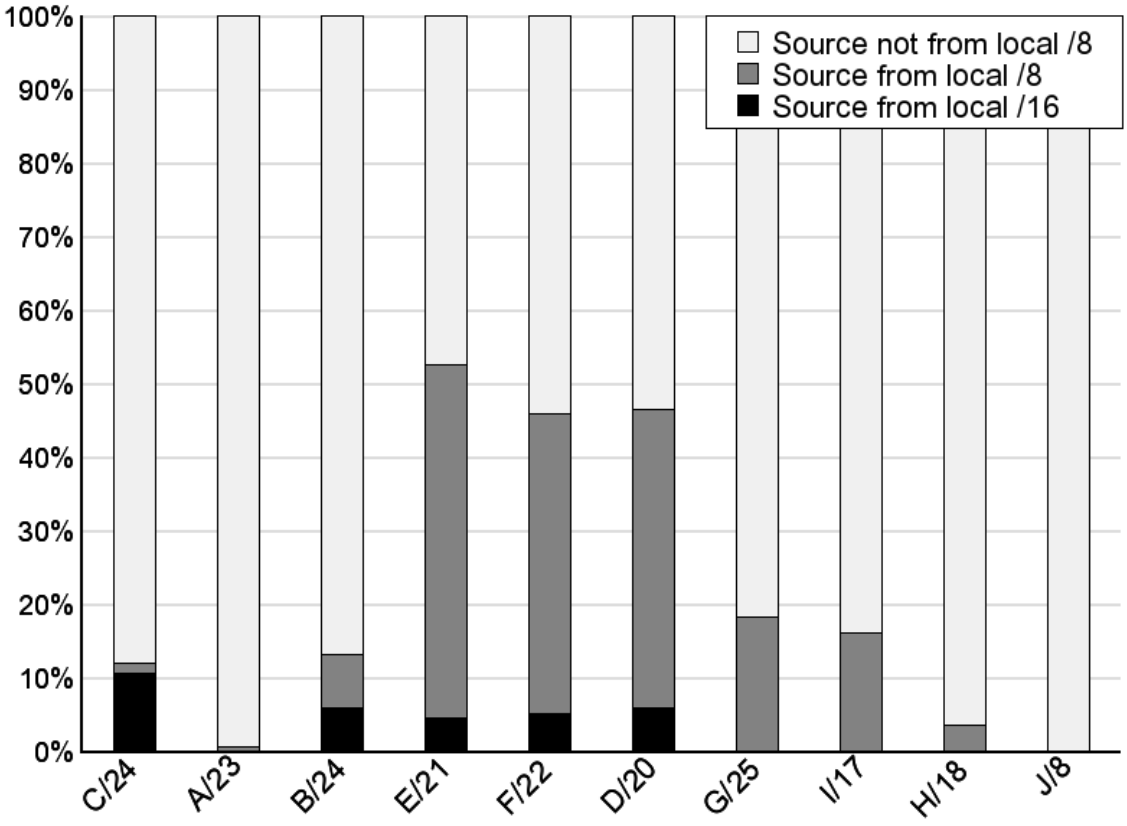


Figure 3.3: Distribution of local preference per sensor.

person or program may prefer to scan local addresses. Take for example someone on a university network attempting to find open file shares. This person will likely scan other blocks in the same /16, assuming these address blocks are also in the university space. Another example is Internet worms that have a local address preference in their scanning algorithms, such as Code Red II [23], Nimda [22], and Blaster [24].

If local traffic was a significant component of the overall traffic, it might explain the differences seen between the sensors. In particular, if the blocks that see the most traffic also receive a large amount of local traffic, that might explain the overall inequalities. Figure 3.3 shows the percentage of traffic to all protocols and services, from the same local /16 and local /8 as the sensor where the traffic was observed. There are two important implications of this graph. First, there are very different relative amounts of local /8 traffic seen at the various sensors. Interestingly, the three blocks belonging to a single ISP (D/20, E/21, F/22) observe close to 50% of local traffic, and the other blocks each see less than 20%. One might expect more local /8 traffic in the academic networks,

Sensor	80	135	137	139	445	1025	4444	Other
A/23	1.2	9.5		1.3	16			4.9
B/24		11		0.2	68		0.7	1.1
C/24		104	0.3		30		0.6	3.9
D/20		4.1		0.7	4.9	0.7		2.2
E/21		5.4		1.0	9.5	0.7		3.1
F/22		4.5		0.9	7.0	0.7		2.4
G/25		1.4	0.2		1.2		0.5	0.6
H/18		0.5		0.1	1.8		0.1	0.4
I/17		1.3		0.1	1.1		0.1	0.5

Table 3.1: Packets (1,000,000s) to the top 4 TCP destination ports, normalized by /24

given less central control, but clearly other organizations also show significant percentages of local traffic. Second, although some sensors see a very significant amount of normalized local /8 traffic, those blocks do not correlate with the sensors with the greatest magnitude of overall traffic. For example, C/24 observes by far the greatest amount of traffic but less than 10% of that traffic is from within the same /8 as the sensor. So, even though local traffic can be significant, the major traffic differences are not due to local preference.

The next step is to break down the traffic by protocol and port. We start by looking at the protocol distribution at each sensor. When the global (non-local /8) traffic is broken down by protocol, almost 99% of that traffic is dominated by TCP. This is logical because a large portion of malware and scans are targeted toward TCP and because the IMS sensors actively respond to TCP packets. One SYN packet response typically generates three or more followup packets, not counting connection retries. Thus, the differences between sensors are dominated by differences in TCP traffic.

Given that TCP traffic dominates at all sensors, the next question is what TCP destination ports are being targeted. Table 3.1 shows the top four TCP ports at each sensor, normalized by a /24. Notice that the distribution is different across sensors and only TCP port 445 and 135 are consistent across all sensors. Despite a preference toward actively responded ports, it is interesting that other TCP ports, like 137 and 139, show up as top ports. The similarity in the top ports across all sensors implies that the differences we see cannot be explained by traffic targeted at ports observed at some sensors but not at others.

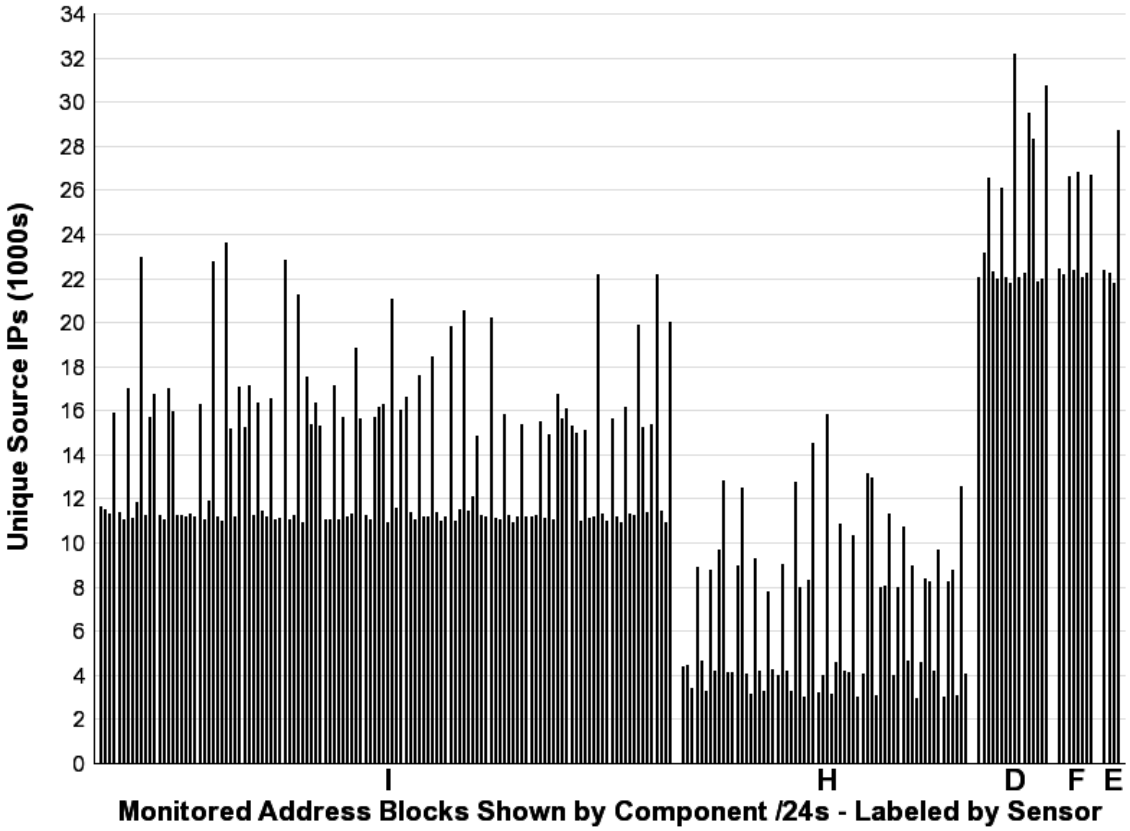


Figure 3.4: Unique source IPs to TCP 135, by /24, excluding local /8 traffic.

3.3.2 TCP port 135

Thus far we have shown that the differences between sensors are not due to a simple port or protocol bias. To the contrary, TCP accounts for almost all traffic and TCP ports 135 and 445 are major contributors to the traffic at all sensors. If these ports account for such a large portion of the overall traffic, an interesting question is whether this traffic is correlated with the differences in overall traffic. TCP port 135 is used by the Windows DCOM RPC service, which has been the target of several vulnerabilities and exploits. In particular, TCP port 135 is the infection vector for the well-known and still prevalent Blaster worm.

Figure 3.4 shows the number of unique source IPs observed on TCP port 135 across the individual /24s that make up the sensor blocks. Each bar in the figure represents the number of unique source IPs seen at a /24. Thus, large address blocks like I/17 in are composed of many component /24s that are all shown individually. Blocks are labeled on the horizontal axis and the separation

between blocks is denoted with a small amount of white space. Sensor blocks smaller then a /22 and the J/8 block are not shown for legibility reasons.

Figure 3.4 has two important implications. First, there are significant differences between the component /24s within the each block. This means that even the destination addresses within a sensor block observe different sources. Second, there are even larger disparities between the unique sources seen across sensors, and these differences correlate with differences in overall traffic (Figure 3.2).

3.3.3 Blaster signature

To test the propagation hypothesis, we isolated a particular signature that has a well known propagation strategy. The signature we chose was that of the Blaster worm. Blaster has a simple propagation mechanism that is based on a sequential scan through IPv4 space. When the Blaster worm is launched due to a new infection or a rebooted computer, the worm chooses an initial target address that is in the same local /16 as the source 40% of the time and a completely random address the other 60%. The Blaster worm then scans sequentially from that initial address, attempting to infect IPs in blocks of 20 at a time.

Figure 3.5 shows Blaster infection attempts by unique source IPs, as seen by a /24. In order to eliminate the possibility of certain blocks being biased by Blaster's local preference, Blaster sources from the same /16 were eliminated. This figure again reveals large differences between sensors. This is a very surprising result. Even though we have attempted to control for propagation strategy, there are still significant differences in the number of unique sources between sensors. Another interesting observation is that the sensor blocks that observed more overall traffic (Figure 3.2) also observed relatively more Blaster sources. However, there are certain hotspots (for example, the large spike in I/17) in the middle of blocks that do not correlate with patterns in overall traffic. The regularity of the distribution indicates some other non-random process may be at work. For example, a poorly-designed random number generator or a bad source of entropy may contribute to the results depicted in Figure 3.5.

This section has demonstrated differences between sensors using three successively more specific views of traffic to a network of distributed blackhole sensors. The first view showed differences in the traffic observed at each sensor over all protocols and services. The second view demonstrated that these differences persist on traffic to TCP port 135. It was also shown that inter-sensor differences dominated intra-sensor differences. The third view established that differences existed even

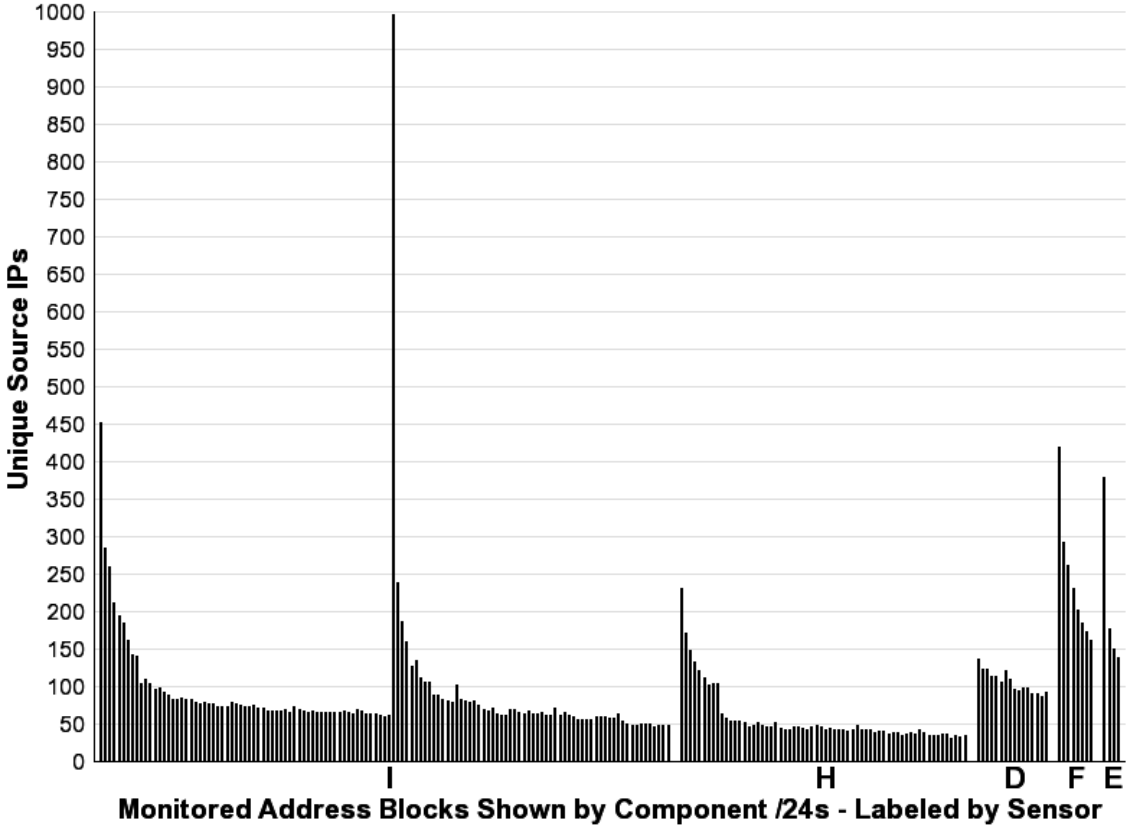


Figure 3.5: Unique Source IPs of Blaster Infection Attempts, by /24, with local /16 traffic removed.

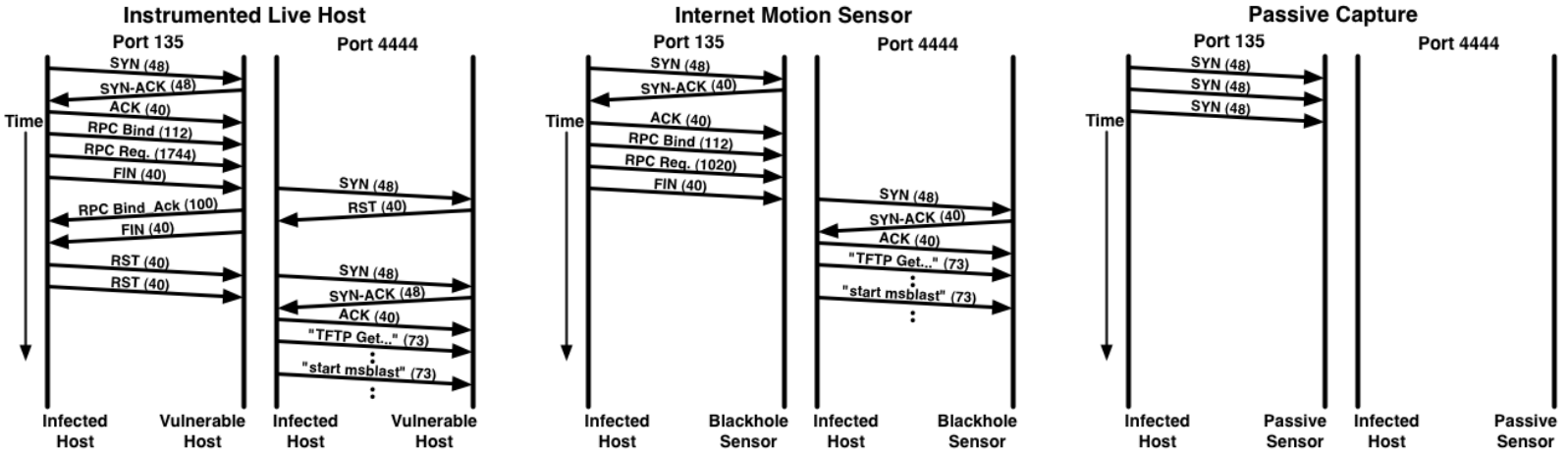
when observing the Blaster worm which, has a known propagation strategy. The result is that there is no definitive explanation for the differences between sensors and more investigation is required. In the next section, we enumerate the distinguishing properties of blackhole sensors in order to better understand how these differences arise.

3.4 Lightweight Responder

Another novel aspect of this work is the construction of a lightweight responder whose main responsibility is to elicit payloads for TCP connections. Recall that because TCP is a connection-oriented protocol [109], no application data is sent until after connection establishment. This has two major repercussions for any TCP based threats: threats to the same port cannot be distinguished from each other, and threats will not send the exploit payload if a connection cannot be established.

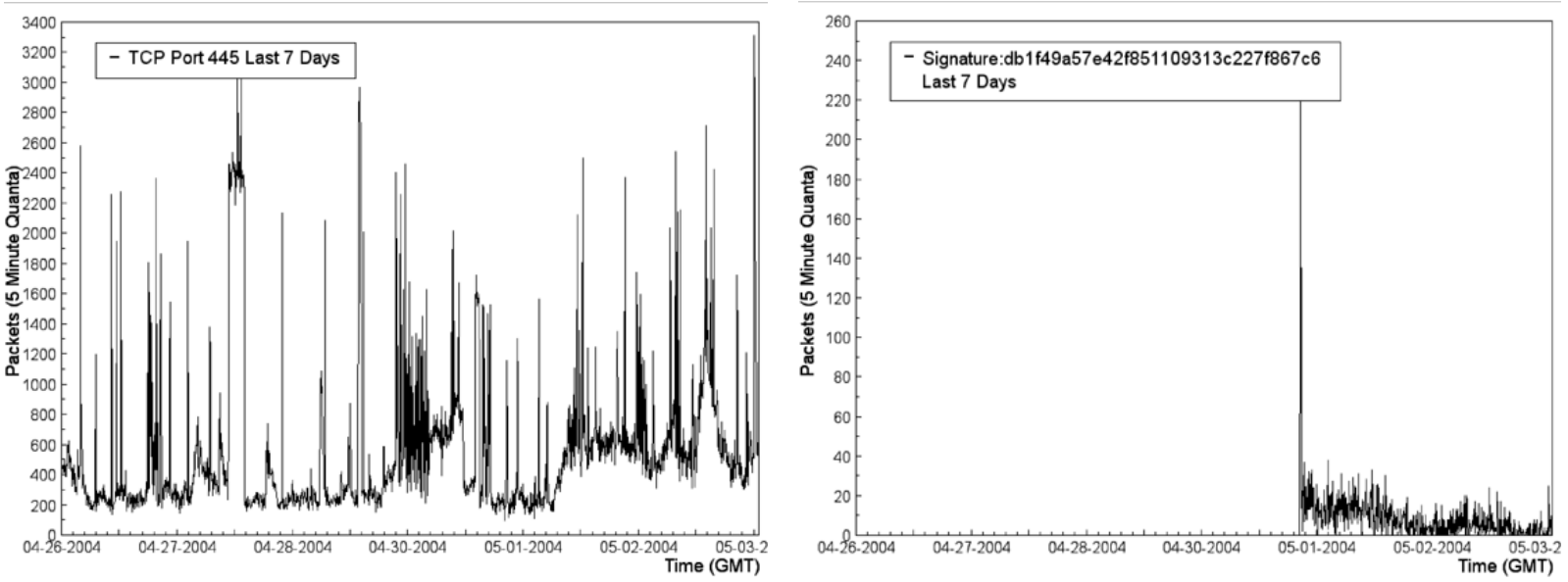
As an illustration, consider the Blaster worm [80]. The infection and transmission method used by Blaster is relatively complicated compared to a single-packet worm like Slammer [76]. A condensed diagram of the transactions involved in a Blaster infection is illustrated in Figure 3.6(a). The Blaster worm first opens a TCP connection to port 135 and sends an RPC bind request. Next, an RPC request message is sent containing a buffer overflow and code to open a backdoor port on TCP port 4444. The newly infected host then sends a message via the new backdoor to download the worm payload and execute it.

Figure 3.6(b) depicts the transactions of the Blaster worm as captured by a blackhole sensor in the IMS. Observe how a single SYN-ACK on port 135 elicits not only the exploit, but also a SYN on the backdoor port. Since the IMS blackhole sensors respond to SYN packets on all ports, the worm connects to port 4444 and sends commands to TFTP the payload and start the worm binary. Compare the data captured by the IMS to the data recorded by a passive blackhole monitor, shown in Figure 3.6(c). While a passive monitor might catch a single packet UDP worm like Sapphire, it will only see SYN traffic from TCP worms like Blaster. The Blaster example illustrates the two key contributions of the lightweight responder, the ability to elicit payloads to differentiate traffic and the ability to get responses across ports without application semantic information. The following subsections explore these points in more depth.



(a) (b) (c)

Figure 3.6: A Blaster infection attempt as observed by three different monitoring techniques.



(a) (b)

Figure 3.7: The Sasser worm as recorded by an IMS /24 blackhole sensor.

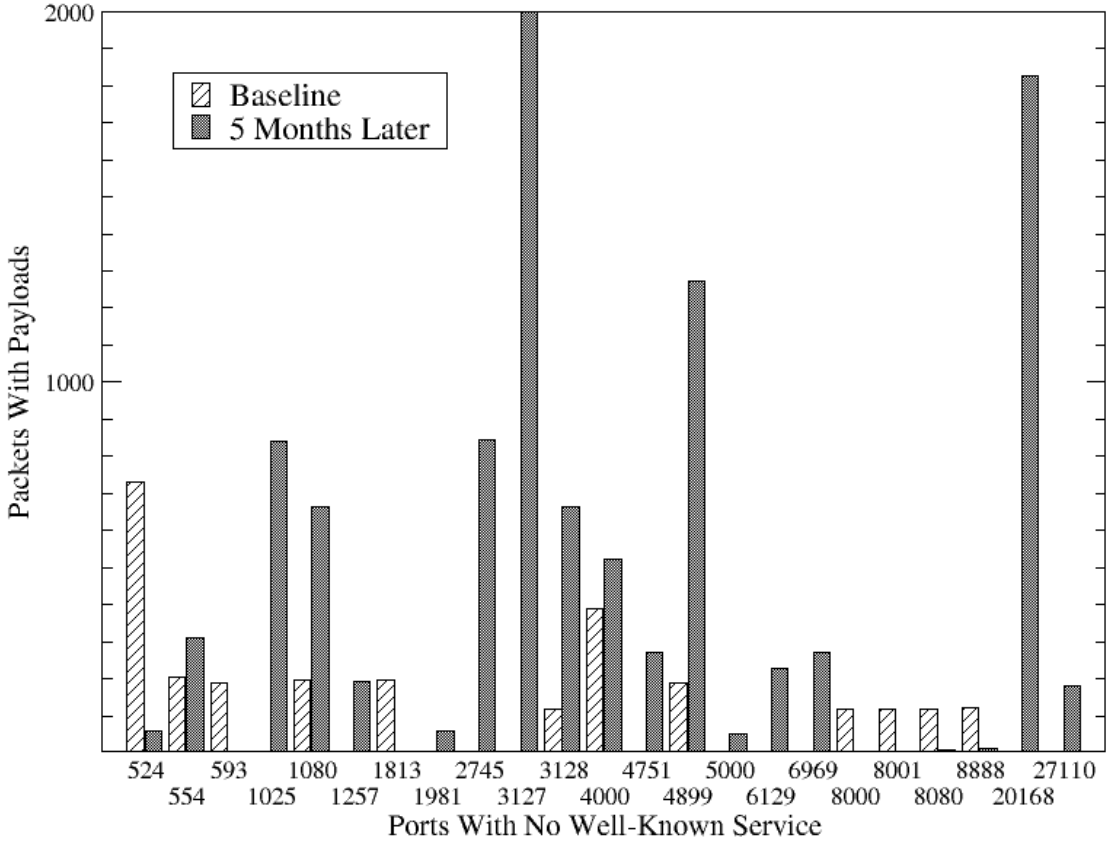


Figure 3.8: Change in activity on TCP ports without well-known services.

3.4.1 Differentiate Services

An important illustration of the value in having more information is the extraction and classification of a new threat for a highly trafficked service. The Sasser [39] worm utilized TCP port 445, which is used by many existing threats. Because the IMS was able to obtain and classify the Sasser payload, it was able to identify the traffic specific to this new worm, shown in Figure 3.7. Figure 3.7 shows the traffic captured on port 445 over the period of seven days. Figure 3.7 shows only the traffic of the signature associated with Sasser over that same time period. Thus, the IMS is able to identify the presence of a new worm, even in an extremely noisy service, by using payload signatures.

3.4.2 Service Agnostic

Another advantage of the lightweight responder is its service agnostic approach, which enables insight into less popular services. Consider, for example, a management application that may not be widely deployed. The population deploying this service might only be several thousand hosts on the global Internet, and the obscurity of the service may mean it is unmonitored. Another example of less well-know services are backdoor ports on existing worms and viruses. New threats that exploit these obscure services can avoid detection by existing network monitoring tools because they do not have the appropriate service modules. In contrast, the IMS can detect and gather significant information on this kind of threat. Consider Figure 3.8, which shows traffic on TCP ports that do not have well-known services. This figure shows the top 20 ports that had significant changes in traffic levels over a five-month period. Note in particular ports 2745 and 3127, which represent services or backdoors as discussed above.

3.4.3 Limitations

The service agnostic lightweight responder is a novel method of tracking emerging threats; however, it may provide little or no information on the threats that depend on application level responses. For example, the NetBIOS service, which runs on Windows systems, requires an RPC *bind()* before performing a RPC *request()* to execute many popular operations. Thus, an IMS sensor may observe the RPC *bind()* but not the subsequent RPC *request()* because no application level response was ever sent. However, in some cases the threat will continue without waiting for the application level response. For example, the Blaster worm will send the RPC *bind()* and RPC *request()* (containing the exploit) without any application level response. In addition, many threats perform operations on multiple ports, and it is possible to track these threats by observing the pattern of requests across ports.

While responses across most or all ports provide insights into potentially unknown threats, we would be remiss if we did not point out that this makes these sensors simple to identify and fingerprint. One main advantage of this system is its focus on globally scoped threats over the activities for individual attackers. This translates into a threat model whereby our biggest concern is the encoding of the monitored network blocks in threat "no-hit-lists." The large number of distributed blocks of small to medium size makes this proposition difficult. We have not seen any evidence to date of this type of activity being encoded into self propagating threats. Nevertheless, we are exploring several

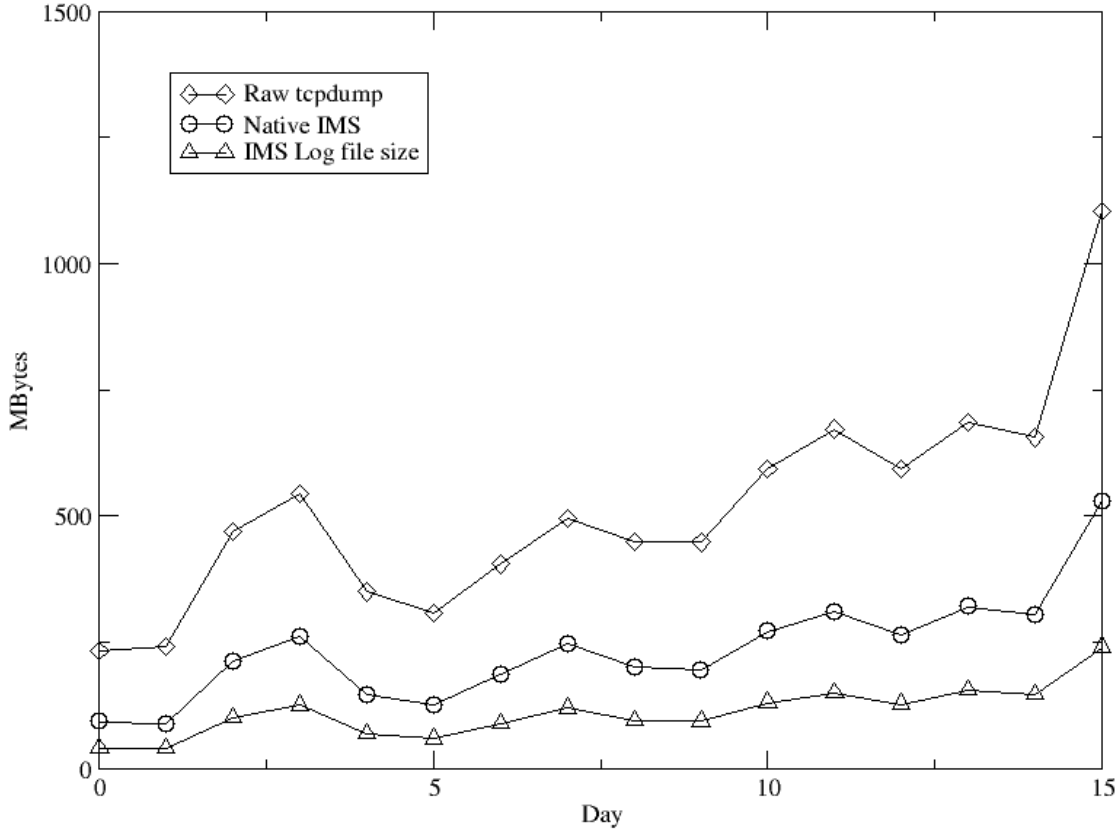


Figure 3.9: IMS and tcpdump log file sizes over a 16-day period.

ways of discouraging fingerprinting including: sensor rotation (i.e., continuously moving the active responders to evade detection), source squelching on individual sensors [82] or across the entire system (i.e., if we detect and respond at one sensor, we don't have to send a response from other sensors), or building simulated hosts and topology (as in honeyd [85]) to mask the presence of a blackhole.

3.5 MD5 Checksuming and Caching of request payloads

The final novel aspect of this system is its method of storing payloads. When a blackhole sensor receives a packet with a payload, it first computes the MD5 checksum of the payload (without network headers) and compares it against the checksum of all the other packets it has seen. If the checksum, or signature, has already been recorded, the passive capture component logs the signature but does not store the payload. If the signature is new, the payload is stored and the

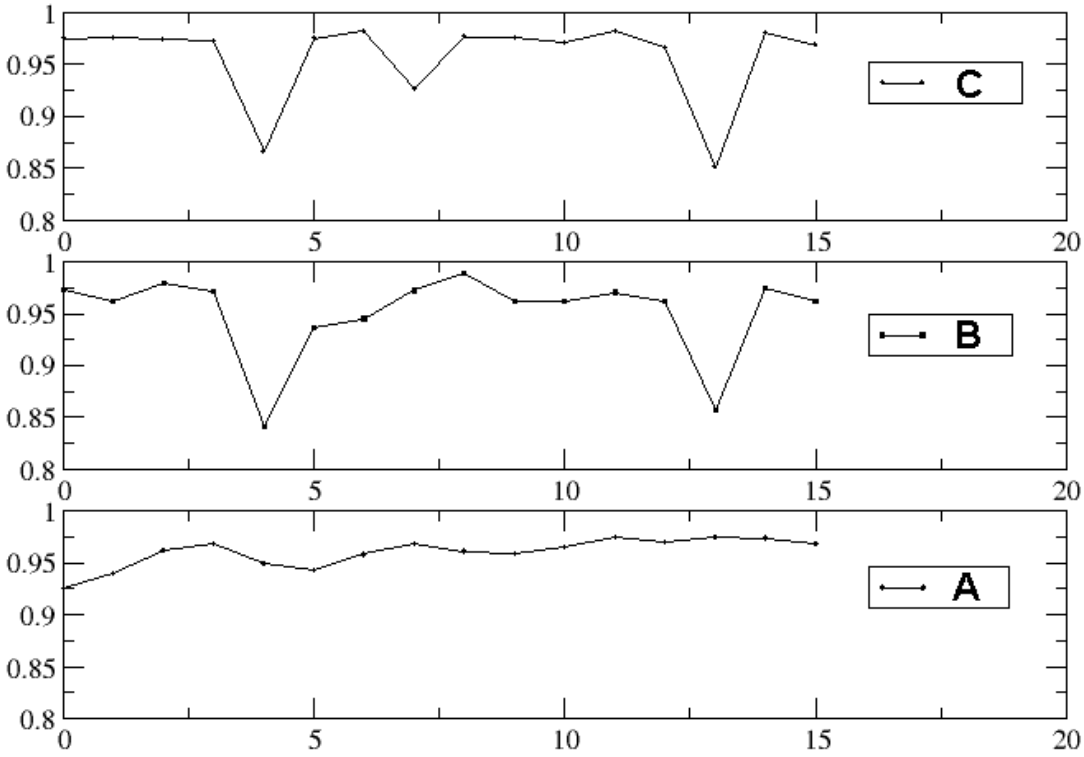


Figure 3.10: Signature database hit rate over a 16-day period on three blackhole sensors.

signature is added to the database of signatures seen. This approach offers a factor of two savings in disk (Figure 3.9), and the hit rate on the signature cache typically tops 96% (Figure 3.10). The implication is that a large number of the payloads seen each day are exactly the same. It is surprising that a cache hit rate of greater than 96% only yields a factor of two increase in savings; however, the payload distribution is heavily skewed to packets with little or no payload.

The factor of two in storage savings helps significantly, since traces gathered on a /8 can reach over 100GB/day. In addition, checksuming every payload provides an efficient signature system. For example, the signatures can be used to generate simple metrics for the amount of new data observed or as a first pass to a more expensive string-matching algorithm. In summary, the MD5 checksuming and caching system in IMS provides a factor of two in storage savings and a simple, fast signature mechanism.

3.6 Summary

This chapter described the Internet Motion Sensor, a distributed, globally-scoped, Internet threat monitoring system. With roots in wide, dark address monitoring, the IMS extends these techniques to include a distributed blackhole network with a lightweight responder and a novel payload signature and caching mechanism. The distributed blackhole network allows increased visibility into more distant events. The distributed sensor component demonstrated an additional important result, namely that significant differences were observed for sensors of equal size, but deployed at different locations. The lightweight responder is designed to differentiate traffic services, while remaining independent of application semantics. Finally, the payload signature and caching mechanism reduces the overhead associated with storing request payloads. Together, these capabilities afford new insight into Internet worms, denial of service attacks, and malicious scan activity.

CHAPTER 4

INTELLIGENT FILTERING IN THE HYBRID ARCHITECTURE

The appeal of hybrid systems is the potential to achieve both the breadth that lightweight network monitoring provides and the detailed forensics of host sensors, without any of the corresponding costs. The key for hybrid systems to be effective at achieving this goal is filtering; they must make intelligent decisions about what darknet traffic to send to the honeyfarm. An ideal mechanism achieves scale by reducing redundant information and forwarding only one instance of each unique threat to the honeyfarm.

In this chapter, we investigate the problem of filtering darknet traffic in order to identify connections worthy of further investigation. In particular, we analyze data from a large, distributed system of darknet monitors. We characterize the traffic seen by these monitors to understand the scalability bounds of a hybrid monitoring system that consists of distributed darknet monitors and a centralized collection of honeypots (or honeyfarm). In addition, we use these characterizations to guide the design of two algorithms that are effective at reducing large traffic rates into a small number of manageable events for the honeyfarm to process. In this chapter, we examine lightweight content prevalence and source prevalence to decide when to hand off a connection.

4.1 Related Work

While any identification of unique threats is imperfect, several techniques from worm detection are applicable. Simple traffic methods use models of normal network behavior to determine when traffic is different from normal. The simplest approach is to watch traffic and alert when that traffic exceeds a certain threshold [94]. The problem with a static threshold is one of configuration and maintenance; that is, a threshold for every network segment is unique and subject to change. Adap-

tive thresholding provides a mechanism for understanding what is normal for a network and setting thresholds [55]. More refined notions of understanding traffic behavior include signal analysis [96], probabilistic approaches [64], and statistical approaches [49]. Content prevalence approaches analyze the distribution of content sequences in payloads to create distributions of content, alerting when a specific piece of content becomes widely prevalent. Two recent works, Autograph [57] and EarlyBird [102], use content prevalence as a mechanism for detection. Both systems use Rabin fingerprints of packet payloads to measure the prevalence of content across a network, alerting or detecting when a piece of content is higher than a threshold.

4.2 Filtering using content prevalence

Content prevalence refers to examining the amount and types of content, or application payload data, that are seen in the network and using the emergence of large amounts of new content as a trigger. Our view of content prevalence differs in several key ways. First, rather than examining content in live networks, we examine content sent to unused networks. This enables us to greatly reduce the amount of packet data that must be processed. This scaling comes from two properties of the darknets: that the darknets see much less traffic than a comparably sized live traffic block, and that much of the malicious traffic repeats itself. This scalability reduction enables our second key differentiator–the ability to examine *every* new payload we see, not just those that are wildly popular.

In this section we investigate the use of content prevalence as a filtering mechanism. We first examine how content can be evaluated and how connections can subsequently be handed off to the host sensor system. We then use data from our IMS deployment to show the effectiveness of this filtering algorithm. Finally, we show how the number of connections generated after filtering is well within the reach of reasonably sized host-based sensor deployments.

4.2.1 Integration of sensors

A potential drawback of content prevalence as a mechanism for filtering interactions to the heavyweight honeypot is that we are unable to determine if a specific connection should be redirected to the host-based honeypots until after a session has been established and content or payload has been received. One solution to this problem is to store sufficient incoming connection information in order to replay the session after a decision has been made. While this is tractable for small

address spaces, a cheaper solution is to only store state for handed-off connections. In order to avoid saving state for every incoming connection, we make the content prevalence decision based off of the first payload packet of any conversation. This requires us to wait for the first payload packet (typically an ACK-PUSH) and use the data in that packet to make the handoff decision. While we must now store state for each connection that is handed off (because we need to rewrite sequence numbers as in [12]), this will cut down on the pace needed.

Figure 4.1 provides an example of this handoff mechanism, as seen in the case of a Blaster worm infection attempt. Four components play a role in this handoff: the attacker who is attempting to exploit a vulnerability, a proxy that decides when to handoff the connection and manages the storing of any state or rewriting of packets, a sensor that characterizes connection attempts and stores payload signatures, and finally, the high-interaction honeypot, labeled "target." In the example, the attacker begins by initiating a TCP connection to port 135. The proxy allows these connection attempts to pass through to the sensor, which ACKs these requests to elicit the payload. The attacker then ACKs the SYN-ACK and immediately follows with a packet containing the application level RPC Bind packet. This first packet with a payload is intercepted by the proxy, and the payload is check summed. If the payload is a new payload, a decision to divert this connection to the host sensors is made. This decision may be made locally, or through coordination with the centralized mechanism that has a global view of the signature cache. A forwarding mechanism is installed and all other packets from that source are diverted to the target. This is important not only for completing the remainder of the existing session, but also to proxy all other connections from the same source. This allows for threats that utilize multiple connections to complete the infection, such as the Blaster worm. Note that because we wish to have consistent data, the lightweight sensor also receives a copy of the RPC Bind payload. Once the decision to divert the stream is made, the proxy must establish a connection with the host sensors using the sequence number information from the RPC Bind. It establishes a connection to the honeypot host and replays the RPC Bind. Note that when the proxy sends the SYN, the target has a sequence number that is unknown to the attacker, and thus the proxy must store state for this proxied connection and must rewrite packets to reflect these new sequence numbers. With this connection established and the proxy rewriting packets, the connection proceeds as normal. The forwarding mechanism divert rule remains in place for that source address for a timeout period so that, as is the case with Blaster, a follow-up connection to another port is possible.

In this section, we discussed several mechanisms for deciding when to perform a connection

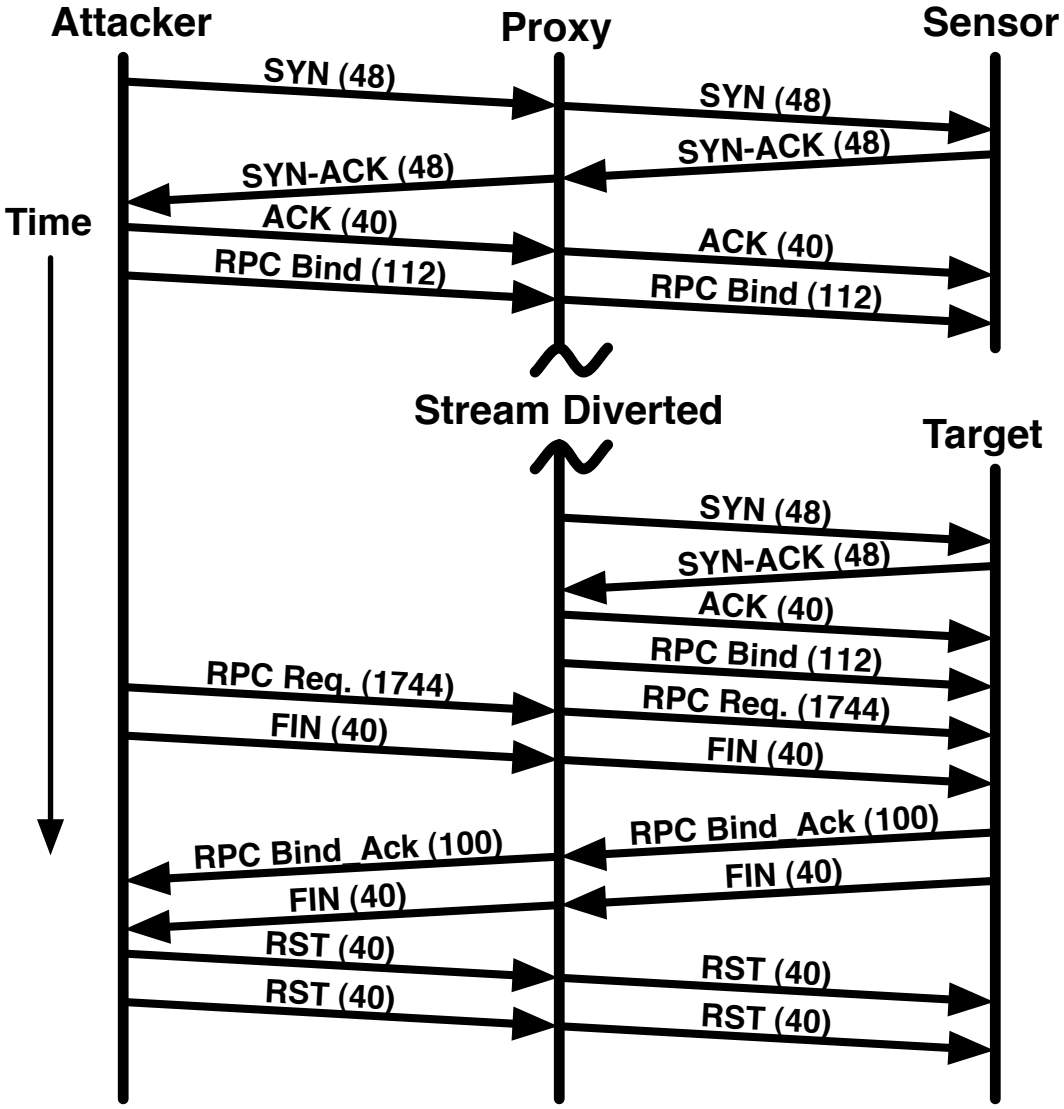


Figure 4.1: The connection handoff mechanism as seen during the Blaster worm.

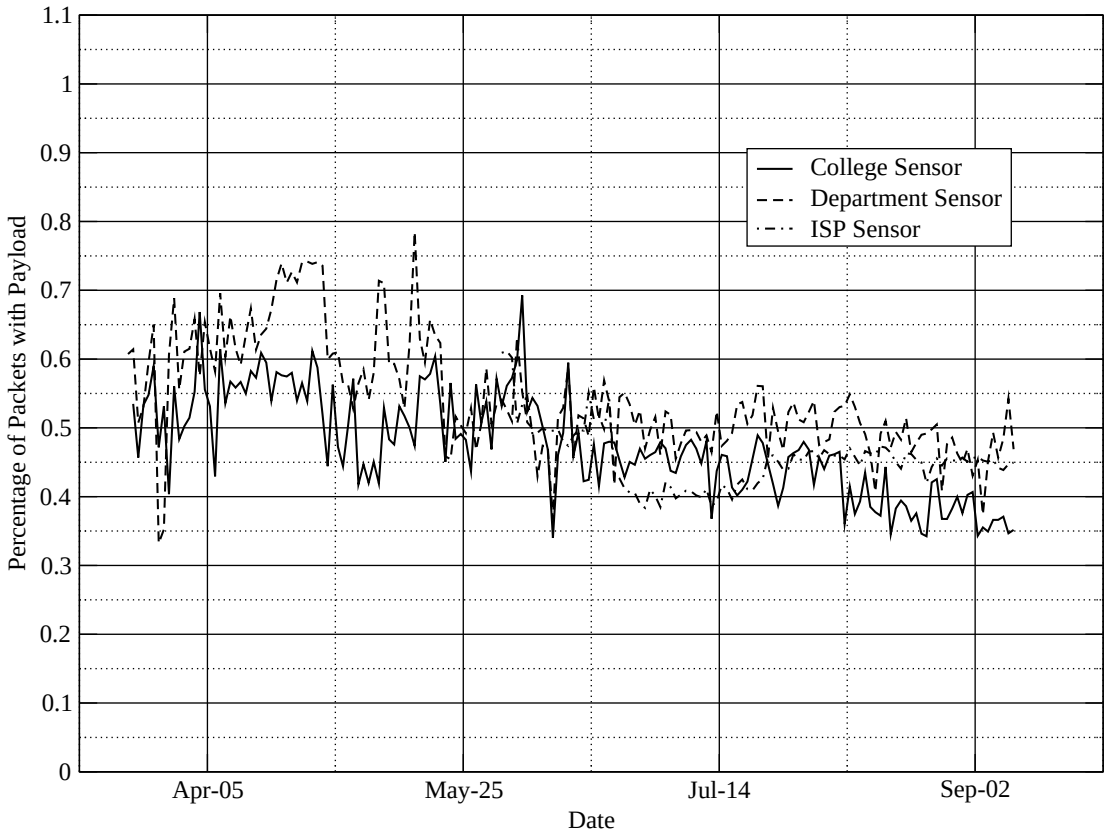


Figure 4.2: The percentage of packets with payloads, as observed by three sensors over a period of six months.

handoff and a specific mechanism, content prevalence. We showed how the need to wait for payload packets complicates connection handoff, and we demonstrated a specific mechanism for accomplishing that task. In the next section, we show the effectiveness of content prevalence in filtering interactions between the lightweight and heavyweight honeypots.

4.2.2 Evaluating the effectiveness of content prevalence

A key to understanding the effectiveness of content prevalence as a filtering mechanism is the realization that many packets observed by a sensor do not contain a payload. These packets may be the result of a scan, backscatter, or part of the three-way handshake. Figure 4.2 shows the percentage of packets with payloads, as observed by three sensors over a period of six months. Roughly 50% of the traffic does not contain a payload. Each of the three sensors observed approximately this same

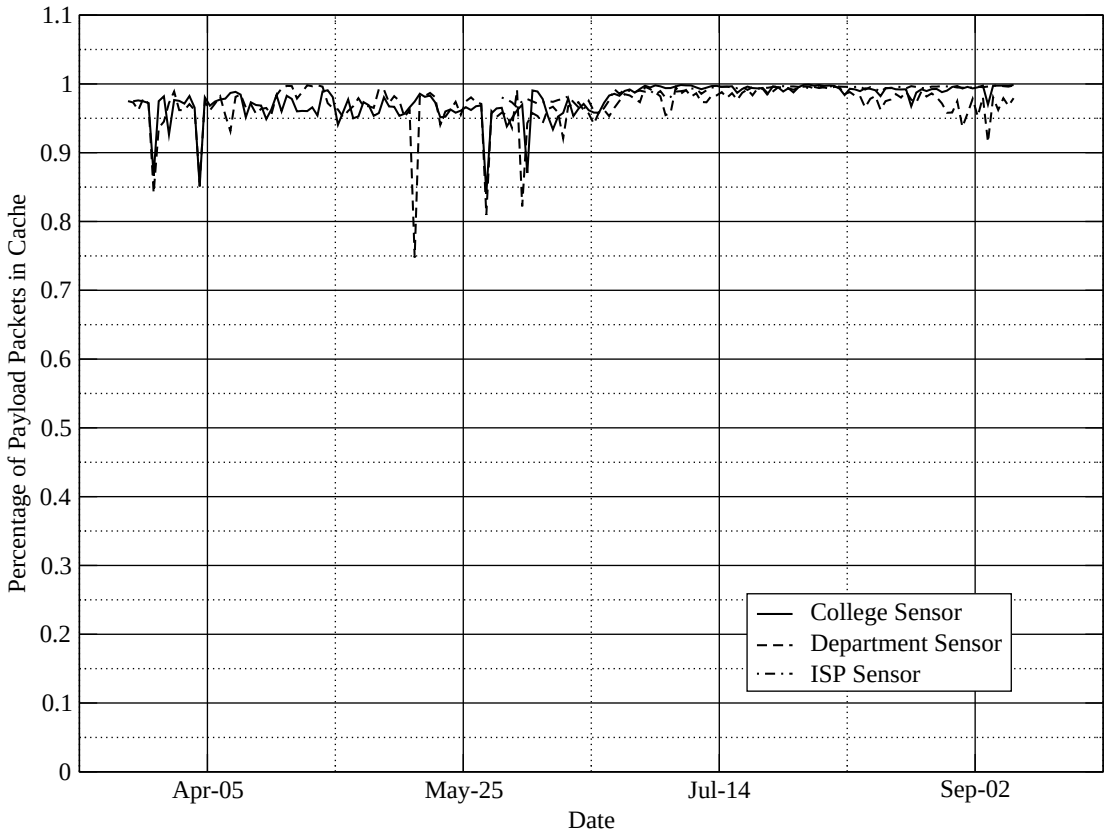


Figure 4.3: The percentage of payload cache hits, as observed by 3 sensors over a period of six months.

percentage of traffic.

One interesting observation made in earlier work is that the contents of the first payload packets are seen many times across blackhole sensors [32]. In fact, roughly 95% of these payloads have been seen before at the same sensor. Figure 4.3 shows the percentage of payloads seen before at the same sensor on the same day, as observed by three sensors over a period of six months. The small number of drops in the rate down to 70% is the result of dictionary attacks where each payload contains a different password from a large dictionary.

A question arises as to whether or not the payloads seen at one sensor are local to that sensor, or are seen globally. Figure 4.4 shows the percentage of payloads that are seen at 0, 1, 2, or 3 sensors over a 5-month period. In the first month and a half, only two sensors are active. In the two sensor case, less than five percent of the packets are cache misses, 35% are cache hits, but only seen at one sensor, and 60% of all payloads are seen at both sensors. When a third sensor is brought online, we

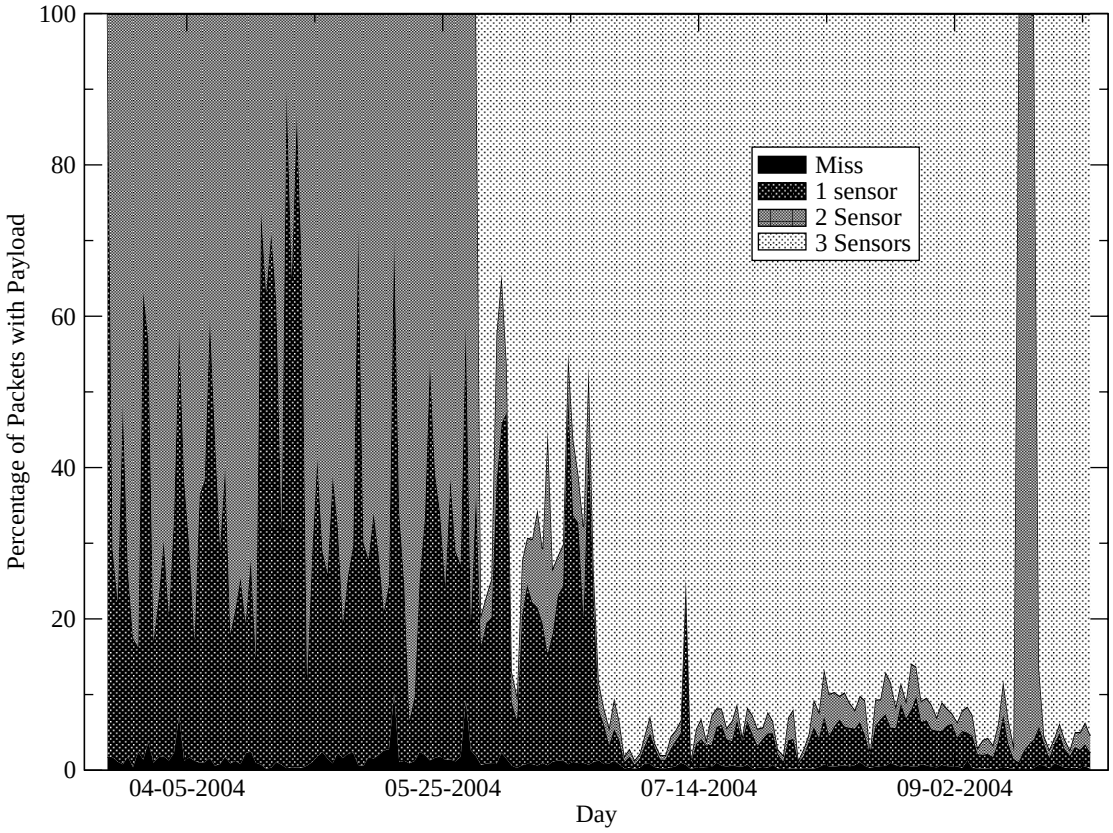


Figure 4.4: The global cache behavior seen at three sensors over a several month period.

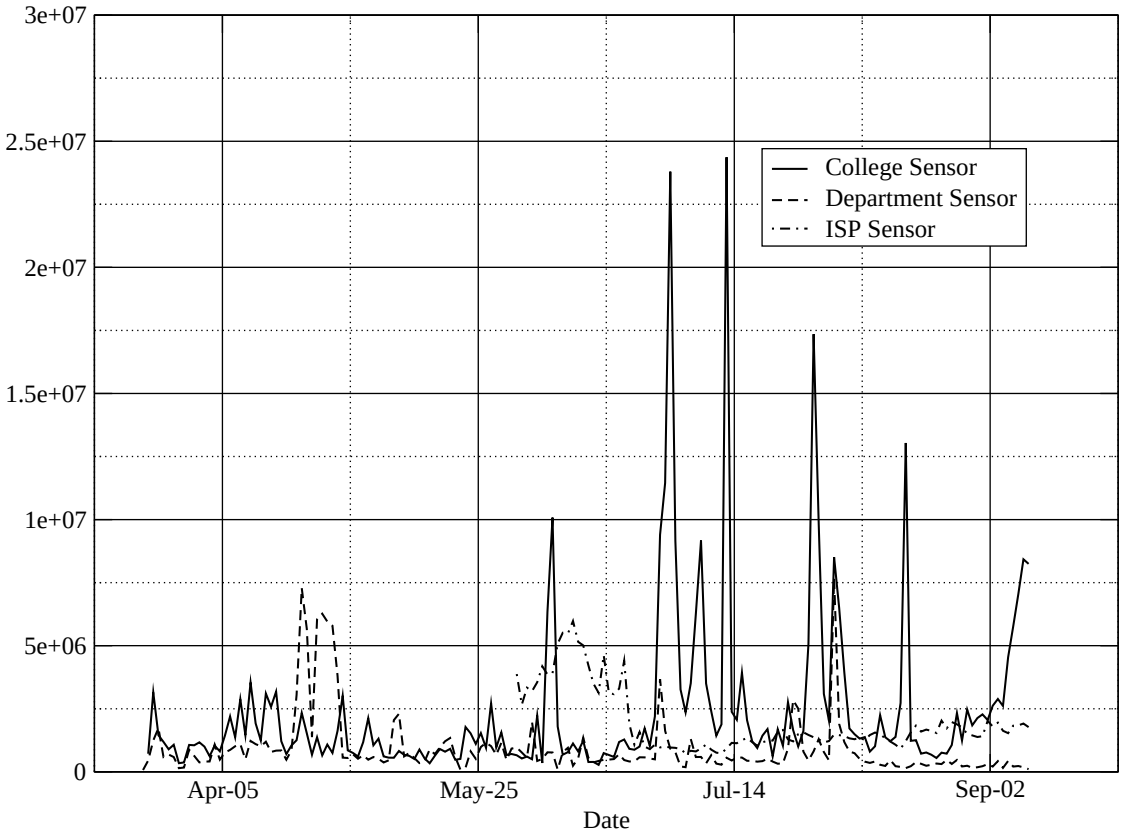


Figure 4.5: The packets seen at three IMS Sensors over a six-month period.

see several months in which over 90% of the packets are seen by all three sensors. This discrepancy is most likely the result of the large packet rate seen by the third sensor, whose distribution of payloads is mostly global in nature.

In this section, we have shown the effectiveness of content prevalence in reducing the candidate set of packets to be considered for host sensor handoff. By observing content as seen at three sensors over a five-month period, we have shown both the percentage of packets that contain no payload, as well as the local and global cache payload rates combined to achieve a significant reduction in the number of connections required by the host sensors. Experimental numbers show that only 0.25% of packets need to be sent to the host sensors.

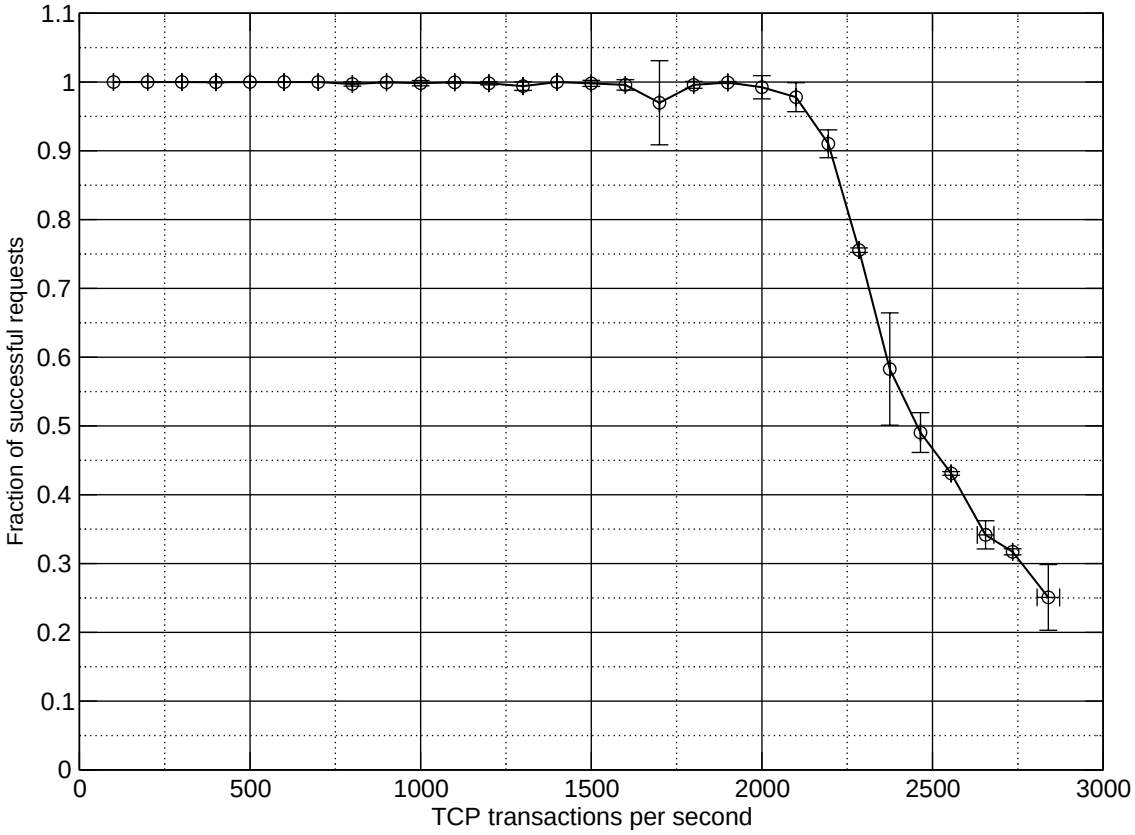


Figure 4.6: The number of TCP transactions per second that Honeyd can support on 1 GHz Pentium III.

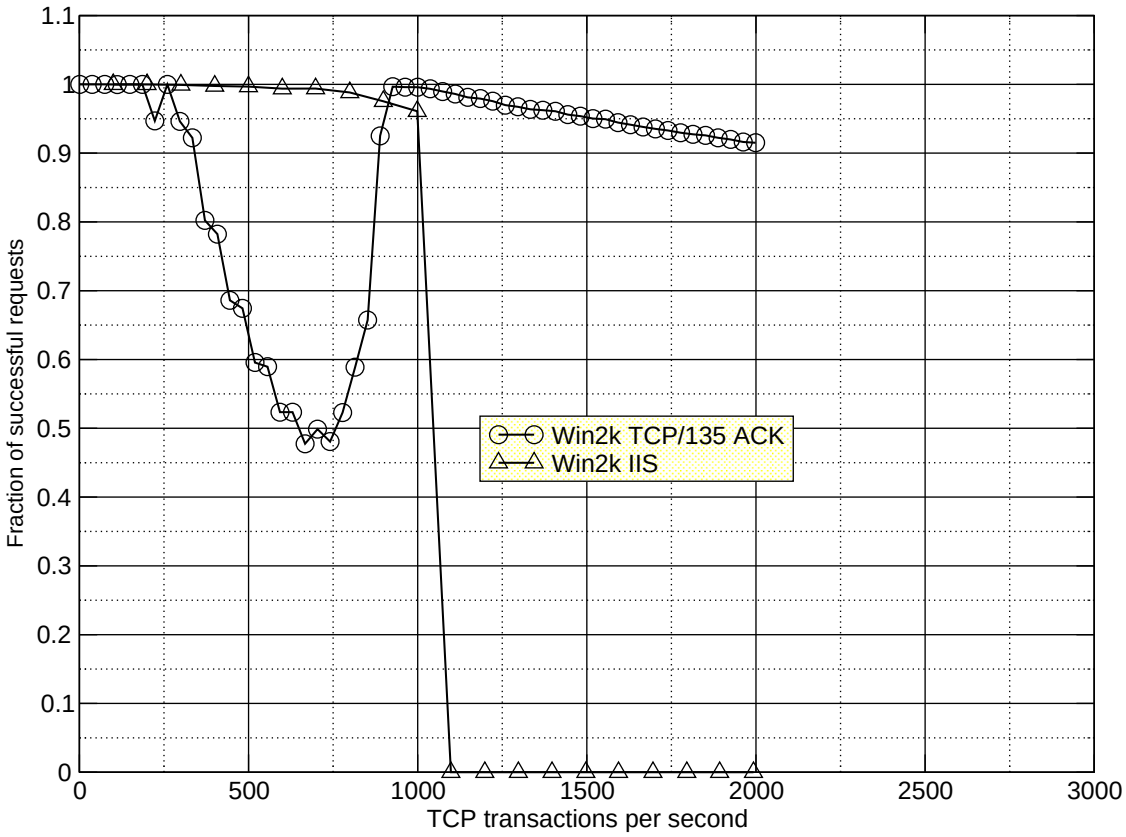


Figure 4.7: The number of TCP transactions per second that Windows 2000 can support.

4.2.3 Estimating system performance

In order to understand the impact of content prevalence as a filtering mechanism, we looked at the workload for each sensor. Figure 4.5 shows the number of packets seen each day across three sensors for a period of five months. It is interesting to note the variations across sensors, both in absolute magnitude and in representation of specific events. The collection of three sensors observes on average 7.5e+06 packets per day, with spikes up to 2.5e+07 packets. This equates to roughly 87 pps across the three /24 sensors. Applying the observations from the previous section, we can calculate the the load generated on the host sensors after filtering based on content prevalence.

$$\text{Connection rate} \times \% \text{ packets w/payload}$$

$$\times \% \text{ payloads new} \times \% \text{ globally new} =$$

$$87 \times .5 \times .05 \times .10 =$$

$$.217 \text{ CPS handed off to the host sensors}$$

This is a drastic reduction in the overall number of connections the host sensors must see. As we will show, easily within the capabilities of both the lightweight and heavyweight honeypots.

Each of the various components of the system must also be able to support the corresponding rates at each level in the filter hierarchy. In order to validate their ability to scale, we ran several laboratory experiments. Figure 4.6 shows the connection per second rate that a typical lightweight honeypot can support. In this case, Honeyd [86] is used as the lightweight mechanism. The system, which is emulating a simple TCP echo response, can scale to over 2,000 Connections per Second (CPS), roughly 10 times the Packet per Second (PPS) rate seen by the collection of three /24 sensors. While each connection contains more than one packet, this is easily within the capabilities of the lightweight mechanism. The host sensors were similarly evaluated in the laboratory environment. We selected a Win2k server for our performance measurement and evaluated the CPS performance of the Windows RPC mechanism (TCP/135) and the IIS Webserver performance, as seen in Figure 4.7. The RPC service starts to show a performance degradation around 300-400 connections per second. The actual increase in performance over time is the result of a Windows SYN flood mechanism that fast paths the TCP 3-way handshake, not passing the connection to "userland" until the connection is complete. The IIS Webserver shows much greater performance, with only a very slight performance degradation until it reaches a 1,000 CPS limit imposed by the client version of Win2k. In any case, a single server is more than able to handle the load placed on it by the three host sensors.

In this section, we combine the filtering benefits of content prevalence with the workload seen at several sensors to show the number of connections per second required by the host sensor framework. We used laboratory experiments to show the scalability of the host-based honeypots as well as the lightweight sensors. Note that the demands created on the host sensors by the three /24 sensors are modest to the point that it is legitimate to suggest that we could replace them with three heavyweight honeypot deployments. However, the intention of the architecture is to scale the monitored address space well beyond three /24 networks, to include millions of monitored addresses. Centralizing the heavyweight honeypots yields more effective utilization of those resources, as well as centralizing those components most at risk of threat propagation.

4.3 Filtering using source prevalence

In the previous section we discussed the use of content prevalence as a mechanism for deciding what to send to the host-based monitors. In this section, we examine the idea of using sources to determine what to analyze. In order to evaluate the impact of source prevalence, we begin by examining the behavior of threats at individual dark address blocks. We then observe these threats across darknets, and based on the insights from these measurements, we construct a filtering algorithm. Then we show how this algorithm is effective at reducing large traffic rates to a small handful of events through a broad production deployment.

4.3.1 Hybrid scalability at individual darknets

For hybrid systems to be effective, they must make intelligent decisions about what darknet traffic to send to the honeyfarm. An idealized mechanism achieves scale by reducing redundant information and only forwarding one instance of each unique threat to the honeyfarm. Unfortunately, the mechanisms for determining what to hand off must make these decisions with the imperfect information available at a darknet monitor. In order to minimize overhead, a darknet monitor typically collects packets passively. As such, it has available elements from the network packet, including the standard five tuple (source IP addresses, source port, destination IP address, destination port, and protocol), as well as any payload data seen for UDP, ICMP, or TCP backscatter packets. As mentioned previously, the IMS darknet sensor also collects the first payload packet for TCP connections through a scalable responder [14]. While other methods have been proposed for eliciting additional application information (for example, by building application responders via Honeyd [86]), in this

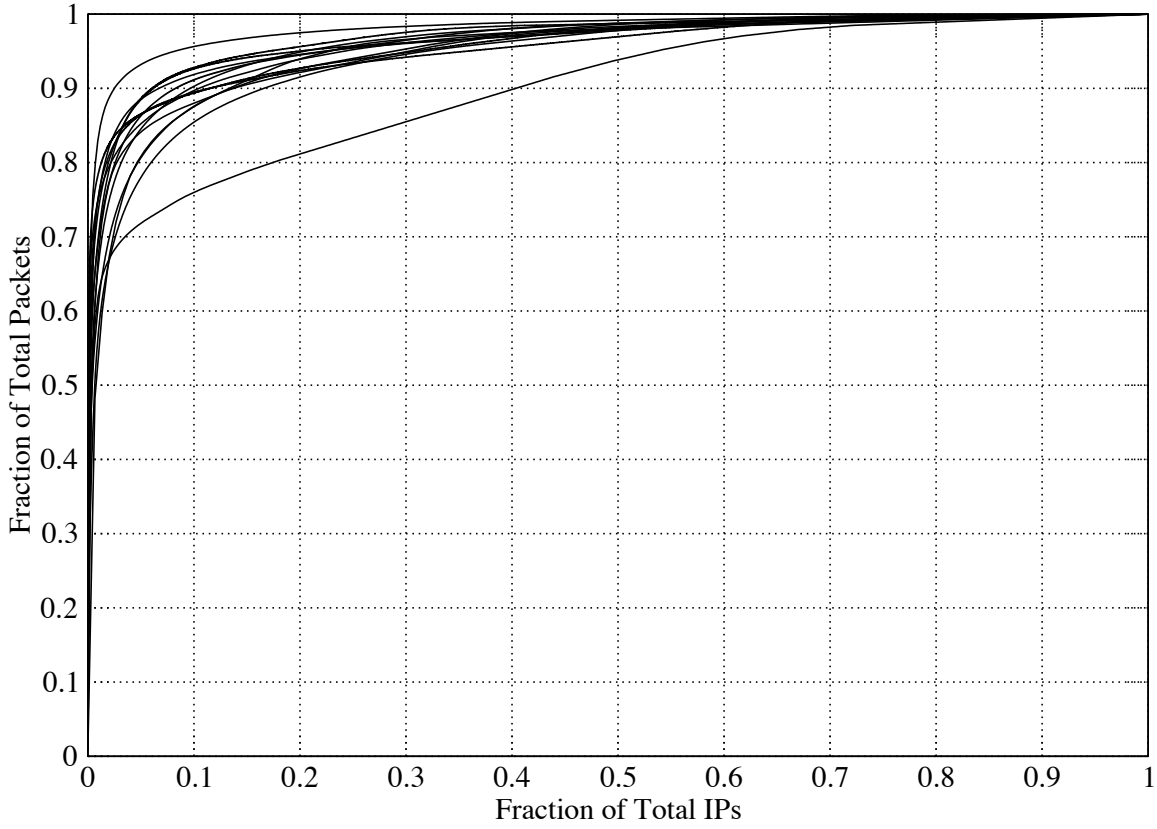


Figure 4.8: The contribution of individual IPs to the total number of packets as seen at 14 darknets.

section we fix the data available for determining what to hand off and leave the issue of exploring more sophisticated information sources for future work.

In the following section, we explore the characteristics of these six elements (the five tuple and initial payload data) in order to determine how they affect the scalability of a hybrid system at individual darknets. We begin by examining the properties of individual darknets and in particular the behavior of source IP addresses. We provide these characterizations by looking at data from 14 darknet monitors ranging in size from a /25 monitor to a /17 monitor over a period of 10 days between August 18, 2004 and August 28, 2004. We then use these characterizations to examine the effectiveness of proposed filtering techniques in reducing the connections that need to be evaluated by the honeyfarm.

4.3.1.1 Characterizing individual blocks

We begin by evaluating the source IP addresses seen at each darknet as a mechanism for determining bounds on the number of unique connections to be evaluated. As with all of the imperfect methods available at a darknet, source IP addresses have limitations in their ability to represent the number of unique attack sources. First, IP addresses do not represent individuals, as multiple users may use the same computer. Second, the mapping from IP address to computer is not static, so a computer may be represented multiple times with different IP addresses [101]. As in other studies [101, 82], we attempt to minimize these effects by performing analysis across small time-frames of less than a day, or more often, less than an hour. However, the actual impact of dynamic addresses is not studied here.

The number of source IP addresses seen at each individual darknet varied greatly, with an inter-block mean of 75,530 sources and a variance of 92,843 sources over the 10 days. The minimum number observed in a single day was 1,345 sources with a maximum of 352,254 sources. Some of the wide variability in sources seen can be attributed to the effect of monitored darknet size. In our sample we had a mean block size of 5,385 IPs (roughly a /22), with the smallest darknet being a /25 and the largest a /17. However, even when normalizing to the smallest block size of /25, we have an inter-block mean of 40,540 sources and a variance of 30,381.

To understand how these source IP addresses behave, we examined the number of packets sent by each source IP address over the 10-day observation period at each of the 14 darknets. Figure 4.8 shows a surprising distribution: over 90% of the total packets observed at each darknet were sent from less than 10% of the source IP addresses seen at that darknet.

The other property that limits the number of unique connections to be evaluated is the number and types of services contacted. To explore this space, we examined the unique destination ports contacted over our 10-day observation period. As with sources, ports are imperfect measures. In particular, destination port analysis suffers from the inability to differentiate activities to the same port and to represent combinations of port activities (as is the case in multi-vector attacks) into a single action. Nevertheless, these serve as a coarse-grained approximation sufficient for exploring the size of the destination service space.

In our analysis, we again see a great deal of variability based on darknet size, with a mean number of contacted ports of 17,780 and a variance of 20,397. The minimum number of ports seen was 1,097 at a /25, and the maximum was 59,960 at a /17. With maximums approaching the total

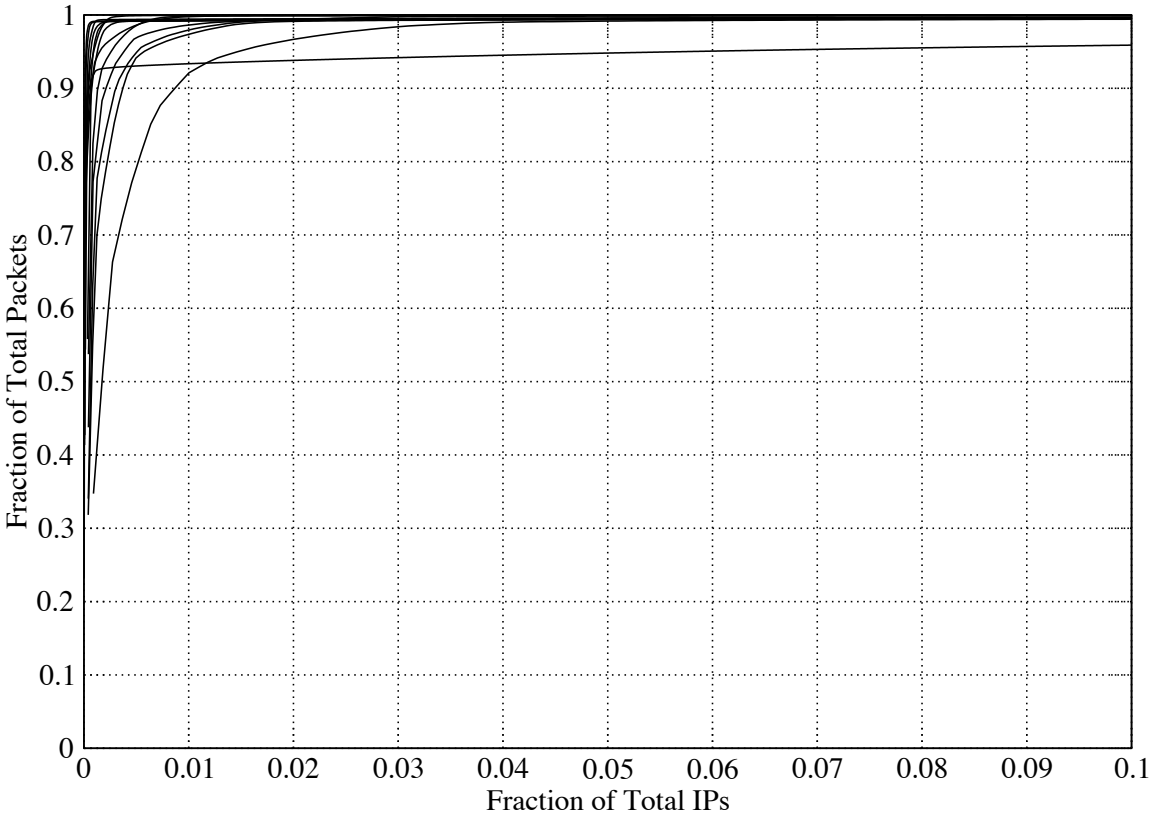


Figure 4.9: The contribution of a port to the total number of packets, as seen at 14 darknets.

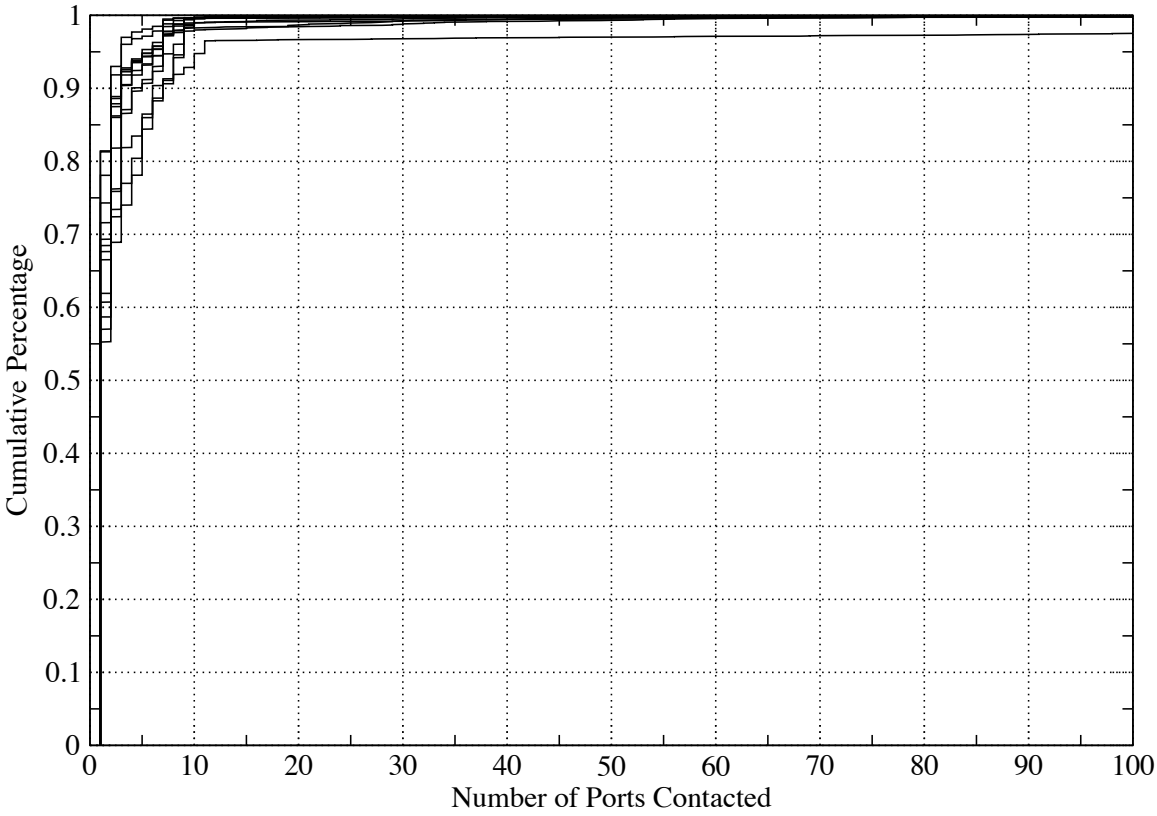


Figure 4.10: The cumulative distribution function of the number of ports contacted for the top 10 percent of IPs seen at 14 darknets.

number of destination ports allowed, we conjecture that many of the ports observed are simply due to ports scans. Nevertheless, unless the scanning is uniform in some way (e.g., sequential) it would be difficult for the darknet monitors to treat these packets differently. To understand the role these diverse destination ports play in darknet traffic, we investigated the distribution of these destination ports and their effect on the number of packets. Again we see a very striking result: over 90% of the packets are from .5% of the destination ports.

Despite the focused distributions, the cross product of the total unique source IP addresses and total destination ports is actually quite large. In order for us to efficiently scale, the source IP addresses must repeat similar actions. We therefore look at the behavior of the top 10% of source IP addresses in terms of the number of destination ports they contact, as well as the number of unique payloads they send. Figure 4.10 shows the number of ports contacted by 10% of the IP addresses at each of 14 darknets over a period of 10 days. At each darknet, over 55% of these source IP addresses contacted a single destination port, 90% contacted less than six, and 99% contacted less than 10.

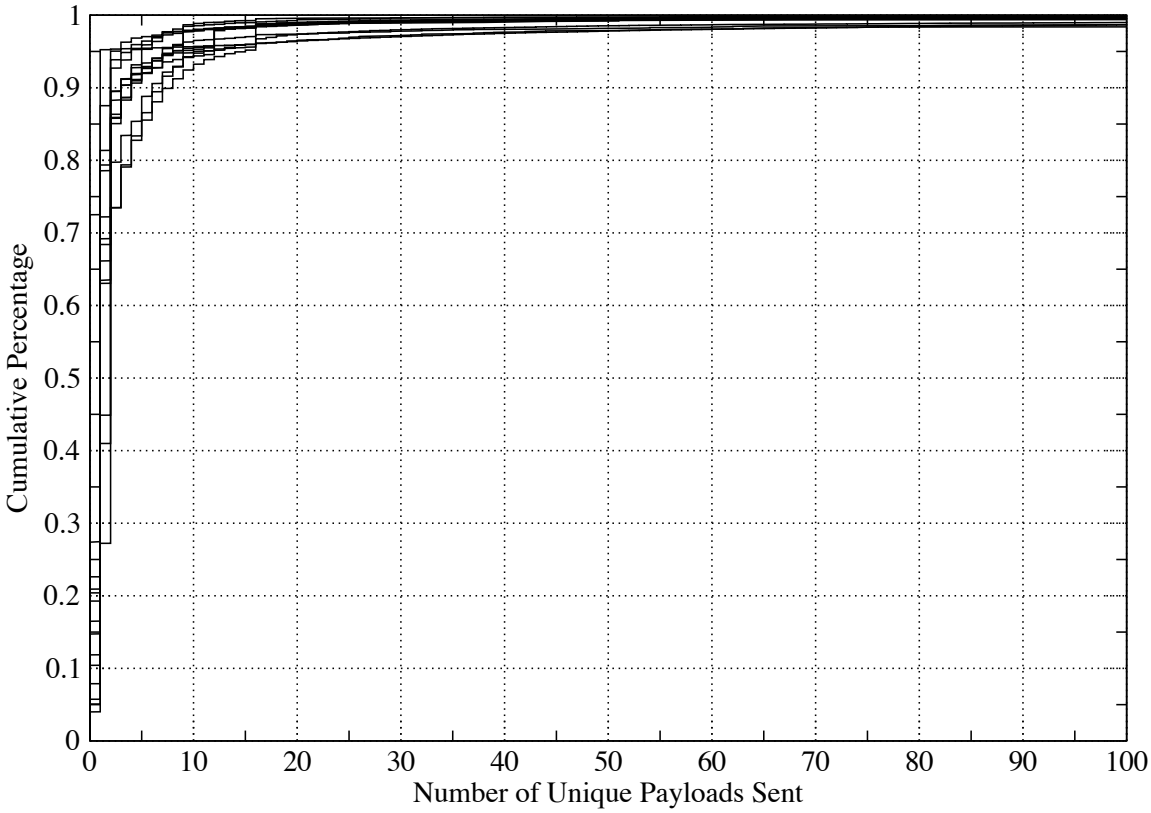


Figure 4.11: The cumulative distribution function of the number of unique payloads sent for the top 10 percent of IPs seen at 14 darknets.

A very small fraction of these very active source IP addresses contacted considerably more ports. As expected, the fanout in the payloads sent is slightly larger, with 30% sending only one payload, 70% sending two or less, 92% sending 10 or less. In this analysis only the first payload of an action is considered. While a better differentiator of threats than ports, it may still under-represent the total number of events, as many services (e.g., Windows RPC) require multiple identical initiation packets. Nevertheless, both methods show that a significant fraction of the behaviors are the same for a source IP address, with the vast majority of the attacks involving multiple destination ports (multi-vector) and consisting of less than 10 contacted destination ports.

In this section we explored the source IP address behavior as seen by 14 individual darknets. We showed that a small fraction of the total source IP addresses is responsible for the vast majority of packets and that these sources are contacting the same, small handful of destination ports repeatedly. In the next section, we examine the effect of these results on several source-based filtering techniques and evaluate the practical value of applying these techniques to reduce packets seen at individual dark address blocks into a smaller number of unique events.

4.3.1.2 Source-based filtering

Recently a small body of work has emerged that exmaines the filtering of the darknet traffic as a means to scale to large address blocks. This work has been published in the context of the iSink [126] project as well as the Internet Motion Sensor [14]. In the iSink work [126], the authors discuss two forms of filtering, random subnet selection and a sample-and-hold method. In subsequent work [82], the authors introduce four types of source-based filtering: source-connection, source-port, source-payload, and source-destination. The tradeoffs are discussed for each, including the effect of multi-stage (multiple connections to the same port) and multi-vector (multiple connections to different ports) based attacks on their accuracy. However, only source-connection is evaluated, and only at two darknets for a two-hour trace. The IMS authors have also discussed [14] preliminary results of using the contents of the first payload packets to reduce disk utilization and to differentiate traffic.

The effect of three of the source-based methods is shown over a 10-day period at 14 darknets in Figure 4.12. In source-connection filtering, N connections from a single source are recorded, and all subsequent traffic from that source is ignored. Source-connection filtering serves as a baseline for determining the maximum reduction in any of the source-based approaches, as each contains the source as a component and is therefore limited to the number of unique sources in a period. In

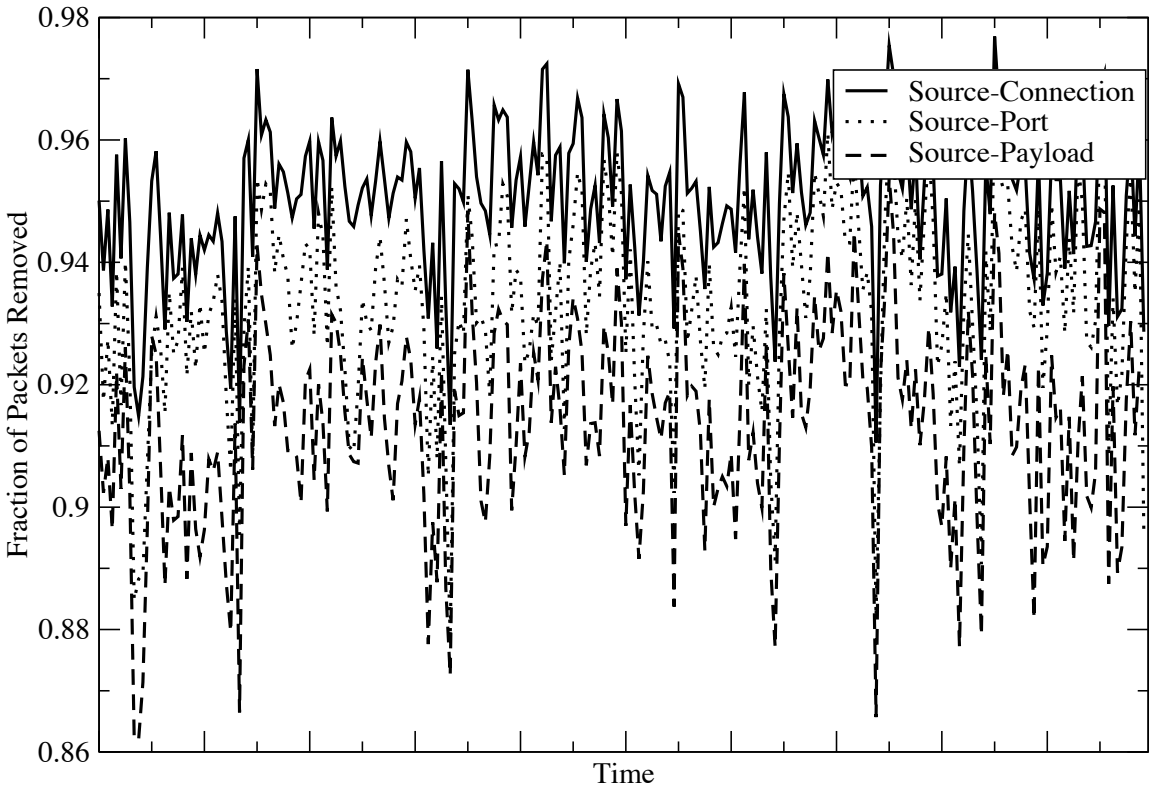


Figure 4.12: The average reduction of source-connection, source-port, and source-payload filtering.

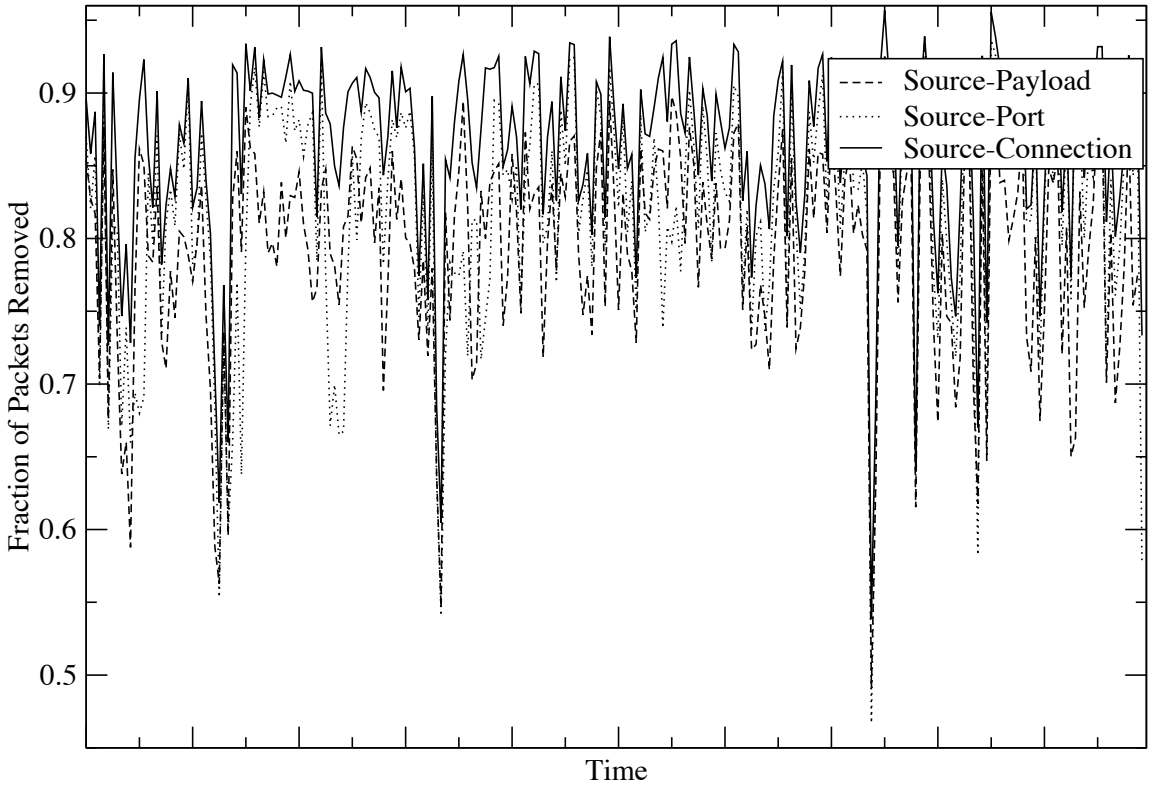


Figure 4.13: The minimum reduction of source-connection, source-port, and source-payload filtering.

source-port filtering, N connections are maintained for every source and destination port pair. This method [82] eliminates the under counting of events of source-connection filtering in the case of multi-vector activities, but multi-stage activities remain a problem, as connections to the same port may not be visible with a small number of N (e.g., N=1 will only record the first connection to a port). Finally, we have source-payload filtering in which the first N payloads sent by a source are used to define all future behavior by that source. We find that the least differentiating view of an event is seen with the source-connection filter. Because it counts any traffic from the same source as the same, it is under counting the number of real events and therefore has the greatest reduction across blocks, a mean of 95%. The more restrictive source-port filtering results in a mean reduction of 93% of the packets. Finally, the most differentiating view of events, source-payload, showed a mean reduction of 91%. On the whole, source-based techniques appear effective at reducing packets to a smaller number of events.

In comparing these results with the source-destination results reported previously [82], we see

less effectiveness than the 99% reduction reported for values of $N = 1$, with even the least restrictive methods. While there are several possible explanations for this discrepancy, such as the difference in darknet block size (the blocks considered in [82] are /16s), we also report a great deal of variance, not only between darknets, but in a darknets over time. The intra-hour standard deviation for source-connection reduction was 2.7%, with source-port and source-payload at 3.1% and 3.9% respectively. Perhaps of more interest is the minimum value during each bin, as this is the run-time value that a hybrid filtering system would have to contend with. We show the minimum values in Figure 4.13. Here the minimum values drop to as low as 53.8% for source-connection filtering, and 46.7% and 49.1% for source-port and source-payload. In practice, the reductions observed may be less when applied to a runtime system that is making decisions about reduction and handoff based on the current observation period.

4.3.1.3 Effects on hybrid scalability

In order to understand the effect of source-based filtering on reducing the number of connections processed by a simple hybrid system, we considered the behavior at a single darknet. We applied source-payload filtering to the largest darknet over our 10-day observation window, a /17 darknet, and counted the unique source-payload events in one second bins. This analysis was designed to measure the total number of events per second a hybrid system consisting of a single darknet would be expected to handle. With averages less than 100, the system can expect to see bursts of several times that, with a single burst above 500 events per second being reported in the 10-day period.

Recall that the purpose of the hybrid system was to identify new threats and receive detailed analysis of those threats. As such, care must be taken to separate the cause of a specific event on the honeypots from its effect (for example, if a honeypot is processing several simultaneous connections, how do we know which one successfully caused the infection?) In the worst case, we may need to separate every event and send it to a separate virtual host for processing. In this case, we now become bound not simply by the event per second rate but also by the duration of the event. To evaluate the cost of event duration, we examined the lengths of connections to an individual honeypot host, as shown in Figure 4.14. While roughly 77% of the connections were of zero length (a connection request and nothing more), the remaining distribution is very heavy tailed, with an average connection length of 400 seconds. In combination, these results indicate that a honeyfarm for a single /17 darknet block would need to handle from 40,000 to 200,000 simultaneous connections.

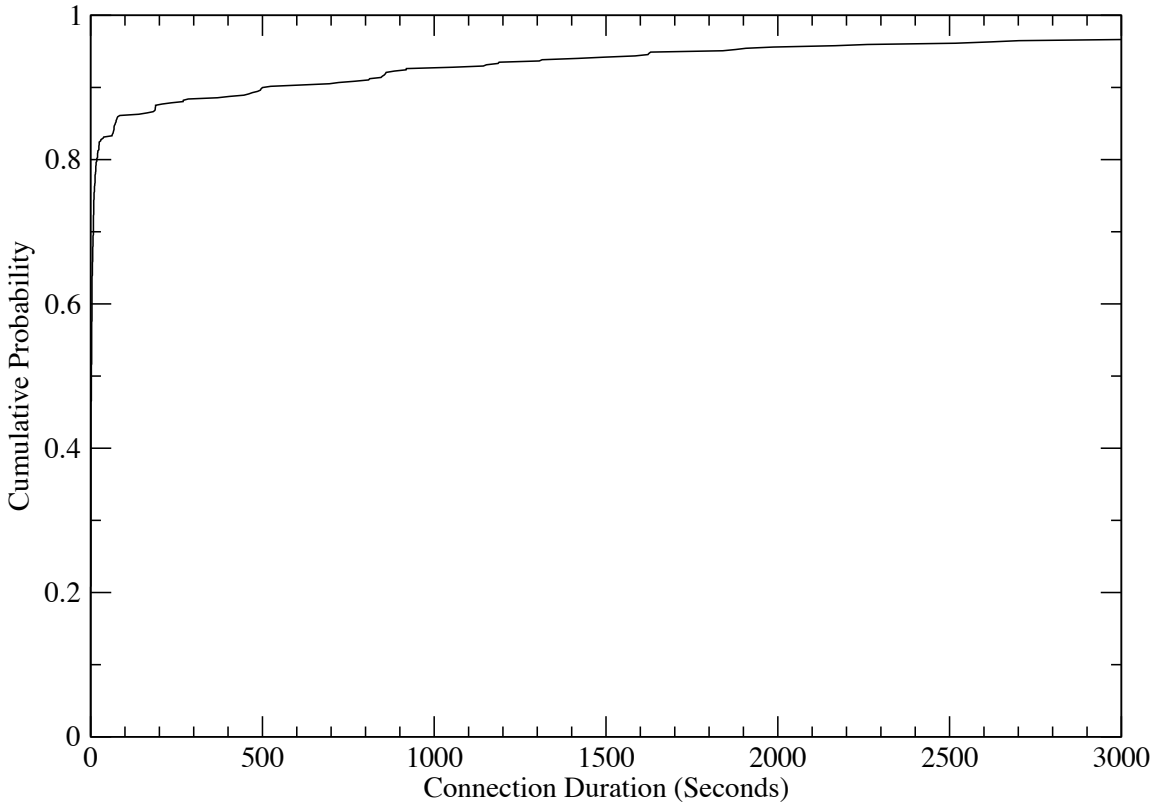


Figure 4.14: The cumulative distribution function of connection length from a Windows 2000 honeypot over a three-day period.

In the previous sections we explored the attack characteristics, as observed by individual darknets including the number of unique source addresses and destination ports. We examined the distribution of packets among these source IP addresses and destination ports and found that a surprisingly small number of source IP addresses, and an even smaller number of destination ports, dominated each darknet. We showed that these distributions make source-based filtering methods very appealing in reducing the traffic at individual blocks, but that there was a great deal of variance in the effectiveness of these methods over time. Nevertheless, we believe that these methods can be very helpful in reducing the number of packets at an individual darknet to a much smaller handful of connections.

4.3.2 Hybrid scalability in distributed dark address blocks

For a hybrid system to be effective, our goals of detecting new threats and providing detailed analysis of these threats must be performed quickly when a new threat emerges. While we showed in the previous section that source-based filters could produce obtainable numbers of connections for handoff, the size of these darknets (e.g., /17) may be to small to provide quick detection of scanning worms. To achieve even further detection time reductions, a darknet monitoring system can choose to monitor a larger darknet, or combine multiple, distributed darknets. In practice however, there is a limit on the size of a darknet (e.g., /8) and few of these large darknets exist. Moving beyond that size when such a large darknet is not available requires additional darknets to be aggregated together. This distributed darknet monitoring approach also has several additional benefits, including added resilience to fingerprinting, and insight into differences between darknets. In this section, we examine the properties of source IP addresses and destination ports across darknets to determine the effects of these properties on a hybrid system consisting of a distributed collection of darknet monitors.

4.3.2.1 Characterizing distributed darknets

We begin by looking at the number of source IP addresses at each of the 41 darknets during a 21-day period, from March 19th through April 9th, 2005. In Figure 4.15, we examine the cumulative unique source IP addresses seen per day. We see that blocks receive traffic from a few hundred (the /25 darknet) to a few million (the /8 darknet) unique source IP addresses per day. There is some overlap with previous days, however, the forward difference still involves hundreds of thousands of hosts every day for the /8 darknet. This order of magnitude difference in the number of source IP

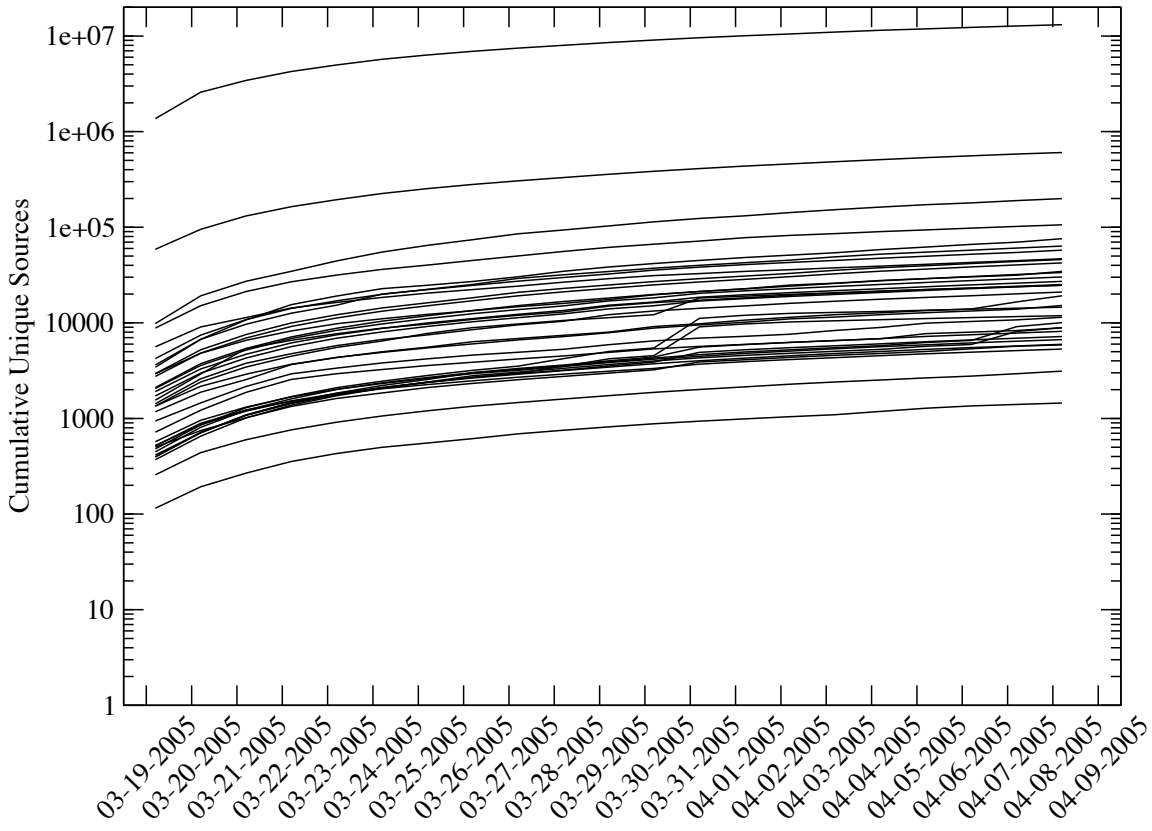


Figure 4.15: The number of cumulative unique sources per day, as viewed by 41 darknets from March 28th, 2005 to April 19th, 2005.

addresses between the /8 and the /17 monitor discussed in the previous section adds a considerably larger number of events to evaluate for the larger darknet.

In order to see how the addition of darknets (each with their own number of unique source IP addresses over time) affects the aggregate number of sources in the hybrid monitor, we computed the overlap in unique source addresses between all darknets. Table 4.1 shows the average percentage of daily source IP address overlap (and standard deviation) in several of the medium to large darknets from the IMS system over a period of one month. A significant number of the source IP addresses seen at these darknets are not globally visible. The largest of the monitored darknets, the /8 darknet, consists mainly of source IP addresses not seen at any of the other darknets, as seen in the D/8 row. However, a much larger fraction of the source IP addresses at the other darknets do appear in the /8 darknet, as seen in the D/8 column. While this implies that many of the source IP address seen at a local darknet are captured by the /8, a significant faction of them are not–from 88% at one of the /22 darknets to 12% at one of the /18 darknets. In addition, the majority of the source IP addresses seen at the /8 are not seen anywhere else. This does not bode well for the scaling of a hybrid system, as the addition of each new darknet will add a significant number of new source IP addresses and hence new connections to be evaluated.

Next we considered the space of the destination ports. For 31 darknets, we examined the top 10 destination ports, based on the number of packets, and compared these lists across darknets. Figure 4.16 shows the number of darknets that had a particular destination port in their top 10 list. The analysis is performed for the top 10 destination ports over a day, top 10 destination ports over a week, and top 10 destination ports over a month. This figure shows that there are over 30 destination ports that appear on at least one darknet's top 10 list. A small handful of these destination ports appear across most darknets (1433, 445, 135, 139), but most of the destination ports appear at less then 10 of the darknets. As with the results seen for source IP addresses, the lack of consistency between darknets implies a broader number of connections to be evaluated, because of the broader number of non-overlapping destination services being contacted.

4.3.2.2 Understanding the overlapping size

The lack of overlap between various darknets in terms of source IP addresses, as well as destination ports, stems from four properties of the monitoring technique and the traffic. These properties are:

	A/18	B/16	C/16	D/23	D/8	E/22	E/23	F/17	G/17	H/17	H/18	H/22	I/20	I/21
A/18	100(0)	25(2)	58(5)	4(0)	78(5)	2(0)	4(0)	55(5)	28(2)	36(3)	28(3)	3(1)	16(2)	12(1)
B/16	23(3)	100(0)	38(5)	3(0)	54(8)	1(0)	3(0)	36(5)	20(3)	25(3)	18(2)	2(0)	10(1)	8(1)
C/16	23(2)	17(1)	100(0)	3(0)	78(6)	0(0)	2(0)	45(4)	20(2)	28(3)	20(1)	1(0)	9(0)	7(0)
D/23	10(0)	10(1)	20(1)	100(0)	30(1)	0(0)	1(0)	20(1)	10(0)	15(0)	11(0)	1(0)	5(0)	4(0)
D/8	2(0)	1(0)	5(0)	0(0)	100(0)	0(0)	0(0)	5(0)	1(0)	3(0)	2(0)	0(0)	0(0)	0(0)
E/22	10(2)	8(1)	13(1)	1(0)	12(1)	100(0)	3(0)	11(1)	9(1)	7(1)	6(0)	1(0)	7(0)	5(0)
E/23	25(4)	20(3)	33(5)	3(0)	34(5)	5(1)	100(0)	30(5)	21(3)	20(3)	16(2)	3(0)	16(2)	13(2)
F/17	23(1)	17(0)	48(1)	3(0)	82(1)	1(0)	2(0)	100(0)	22(0)	29(1)	21(0)	1(0)	9(0)	7(0)
G/18	20(1)	16(1)	36(1)	3(0)	51(2)	1(0)	2(0)	38(1)	100(0)	24(1)	16(1)	2(0)	9(0)	7(0)
H/17	16(0)	12(0)	31(1)	2(0)	53(1)	0(0)	1(0)	31(1)	14(0)	100(0)	27(2)	1(0)	8(0)	6(0)
H/18	20(1)	15(0)	37(2)	3(0)	56(2)	1(0)	2(0)	37(2)	16(0)	45(2)	100(0)	2(0)	11(0)	8(0)
H/22	7(2)	5(0)	9(1)	1(0)	16(3)	0(0)	1(0)	9(0)	6(0)	8(0)	6(1)	100(0)	3(0)	2(0)
I/20	11(1)	8(0)	16(1)	1(0)	16(1)	1(0)	1(0)	16(1)	8(0)	12(1)	10(0)	0(0)	100(0)	14(2)
I/21	13(1)	10(1)	20(1)	1(0)	19(1)	1(0)	2(0)	19(1)	11(1)	16(1)	12(1)	1(0)	21(4)	100(0)

Table 4.1: The average (and stddev) **percentage** of overlap in source IP addresses between (row, column) medium to large darknets over a month.

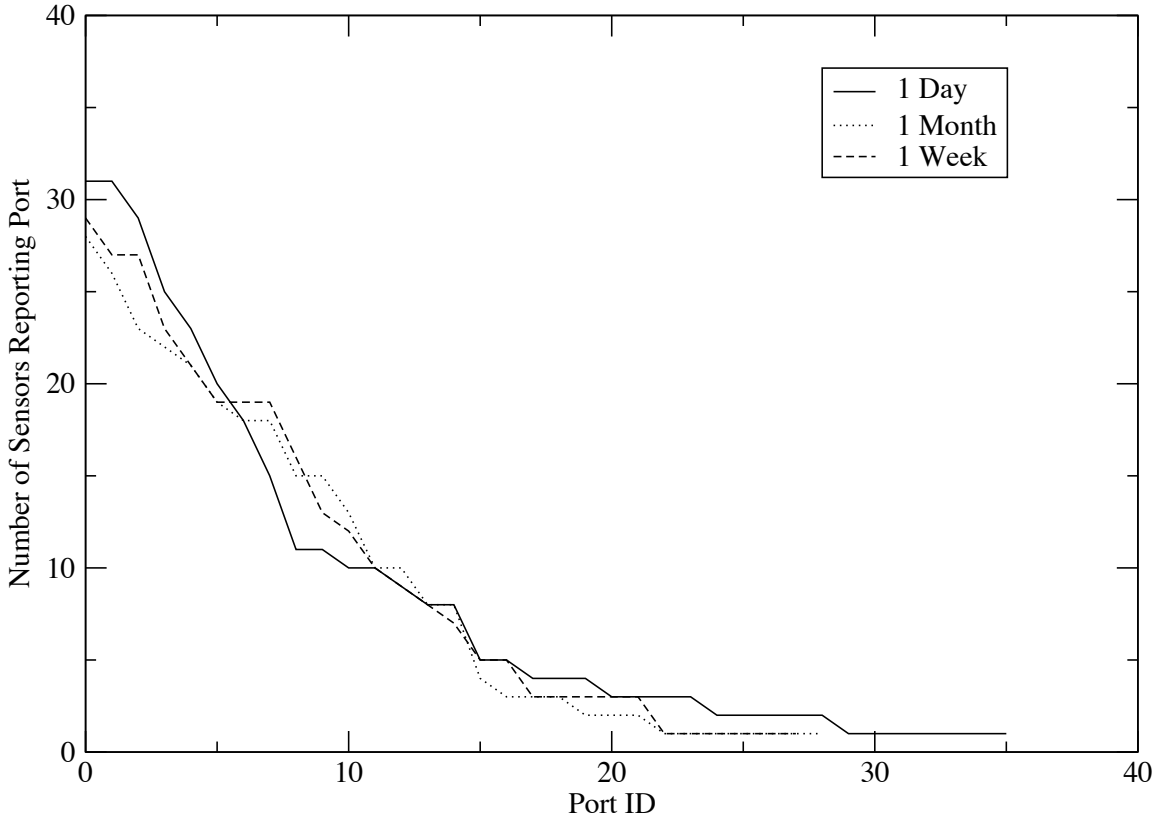


Figure 4.16: The number of darknets (of 31) reporting a port in the top 10 ports over a day, week, and month time frame.

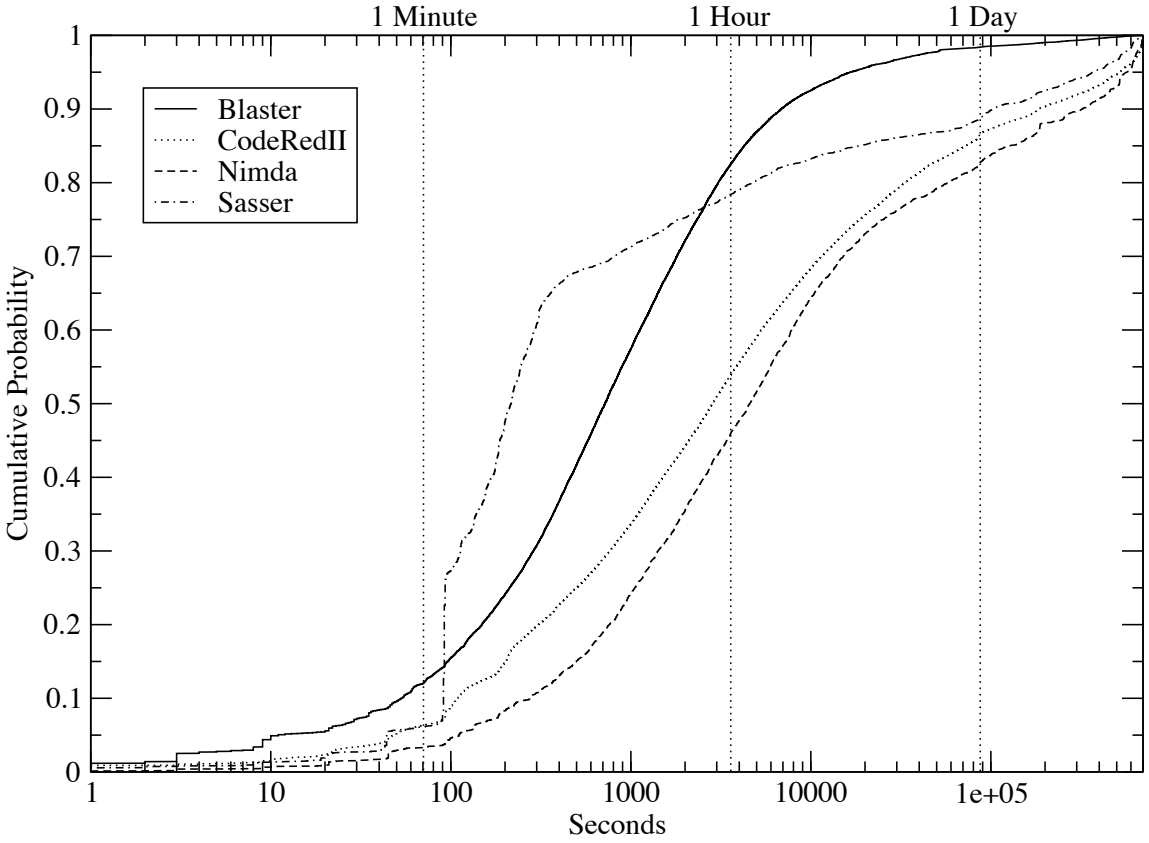


Figure 4.17: The duration of source IP address observations at the /8 darknet over a one-week period for four known worms.

- **The impact of *monitored block size, scanning rate*, and *observation time* on the probability of identifying a random scanning event.** The effect of monitoring block size on the detection time of remote events is a well-studied phenomena [77]. Table 4.1 does show larger blocks with overlapping source IP addresses. However, the largest of these, although highly variable, only sees an average of 50% overlap.

- **The lifetime of the events.** One possible explanation for this lack of overlap is that the events being observed are too short lived. To examine this, we looked at the behavior of four familiar worms whose (random and non-random) scanning behaviors are well known. Figure 4.17 shows these results. We recorded the observation lifetime of the source IP addresses across our /8 darknet. It should be noted that this method can only approximate the actual lifetime, as the /8 darknet may only observe part of the worm's propagation behavior. With this caveat in mind, we note a significant faction of the source IP addresses seen at this darknet had a lifetime of less than one day. However, more than 50% had lifetimes of over an hour, which is sufficient at most scanning rates (greater than 10 connections per second) to be observed at the bigger darknets.

- **Targeted attacks.** While the above explanations provide rationale for some of the lack of overlap, it is important to note a significant number of source IPs do not overlap. It is our conjecture that these source IPs are representative of targeted behaviors and attacks. While this conjecture is difficult to validate without monitoring at the sources of these attacks, we can look at the distributions of destination ports for some insight as to whether the same services are being contacted across darknets. Recall that Figure 4.16 showed the number of darknets that report a port in their top 10, over increasing time frames of a day, week, and month. The results show that, although there are a few ports that are globally prevalent, a large number of the ports are seen over these time frames at very few blocks.

- **Environmental factors.** In [32] we showed that a variety of factors, including filtering policies, propagation strategy, darknet visibility, and resource constraints, affect the mix of the global traffic a darknet should see. These properties provide subtle influences beyond the impact of the targeted behavior discussed above.

4.3.2.3 Effects on hybrid scalability

One of the drawbacks of any of the source-based techniques discussed in the previous section is their reliance on the source IP address as part of the filter. For a distributed technique based on these methods to be effective, source IP addresses must be prevalent across darknets. Unfortunately, our analysis showed this not to be the case. In order to quantify the effectiveness of source-based methods across darknets, we considered the same 14 darknets as in the previous section. We chose source-port filtering and looked at the reduction in unique (source, port) pairs across darknets over the same 10-day period. While this method was effective at reducing the packet space by 95%, there is only a modest 20% reduction when these methods are applied across darknets.

In this section, we examined the properties of distributed darknets to understand the scaling properties of a hybrid system. We evaluated the properties of source IP addresses and destination ports across darknets and observed a significant number of non-overlapping destination ports and source IP addresses. While some of these differences are the result of well-known size and rate effects of darknet monitoring, other factors, such as the short on-time of random scanning hosts and the targeted nature of attacks observed at these darknets, help to explain the additional differences. The impact of these non-intersecting events is that each individual new darknet added to a hybrid system is likely to significantly increase the total number of connections that need to be evaluated. If a hybrid system is to scale in this type of environment, a new definition of events and new methods of filtering are required.

4.3.3 Aggressive distributed filtering

In the previous sections, we discussed source-based filtering at individual darknets and the application of those filtering techniques to distributed darknets for the purpose of building a scalable hybrid system. We noted that source-based techniques were much less effective across darknets for two main reasons: source IP addresses are not typically visible across darknets in time frames that are useful and many attacks may be targeted at individual darknets only.

In this section, we examine techniques for filtering that explicitly account for these observations. In particular, we examine the number of unique source IP addresses per destination port, and we examine the number of darknets reporting an event. We combine these variables with the technique of threshold-based alerting to provide a filter that passes traffic on global increases in the number of source IP addresses contacting a destination port.

Alerting on traffic changes by observing traffic to live hosts is a well-studied area. Existing methods have alerted once traffic has reached a static threshold [94] or for thresholds that were adapted dynamically [55]. Further attempts to model traffic behavior that can be used for detecting changes include signal analysis [96], probabilistic approaches [64], and statistical approaches [49]. In this section, we extend the existing techniques to look at adaptive threshold-based alerting for traffic to unused address space. Similar in vein to other source address distribution alerting mechanisms [124], we watch the distribution of unique source IP addresses to a specific port. However, our work differs from these alerting mechanisms in several key ways: we modify the algorithm to explicitly allow for varying notions of current and historic behavior, we modify it to watch the distributions across multiple darknets, and we incorporate the notion of global versus local behavior.

Our current event identification scheme uses an adaptive threshold mechanism to detect new events. Every hour, each darknet is queried for the number of unique source IP addresses that have contacted it with destination port x (where x ranges over a list of destination ports we are monitoring). The event identification algorithm looks at the collected data and works as follows for each destination port x. For each hour, it adds up the number of unique source IP addresses contacting destination port x at each darknet. It then scans over this data, one hour at a time, comparing the average (per hour) over the event window (last event window hours) to the average over the history window (last event window $\times$ history factor) hours. If the ratio of event window average to history average is greater than the event threshold, an event is generated. These events are then filtered based on whether they are global or local, via the coverage threshold. The coverage threshold defines the number of darknets that would have generated an event individually for a threat. Events will not be generated more than once in a single event window. To protect against false positives on destination ports that have little or no traffic destined to them, whenever the sum of unique source IP addresses from all blocks to a specific destination port is less than one, the data point for that hour is set to 0.6. The event generation algorithm then is parameterized by four variables: the event window, the history window, the event threshold, and the coverage.

4.3.4 Filtering evaluation and deployment results

In this section, we investigate the parameter space of the event identification algorithm and then evaluate it by comparing both the security events we identify and those identified by the community.

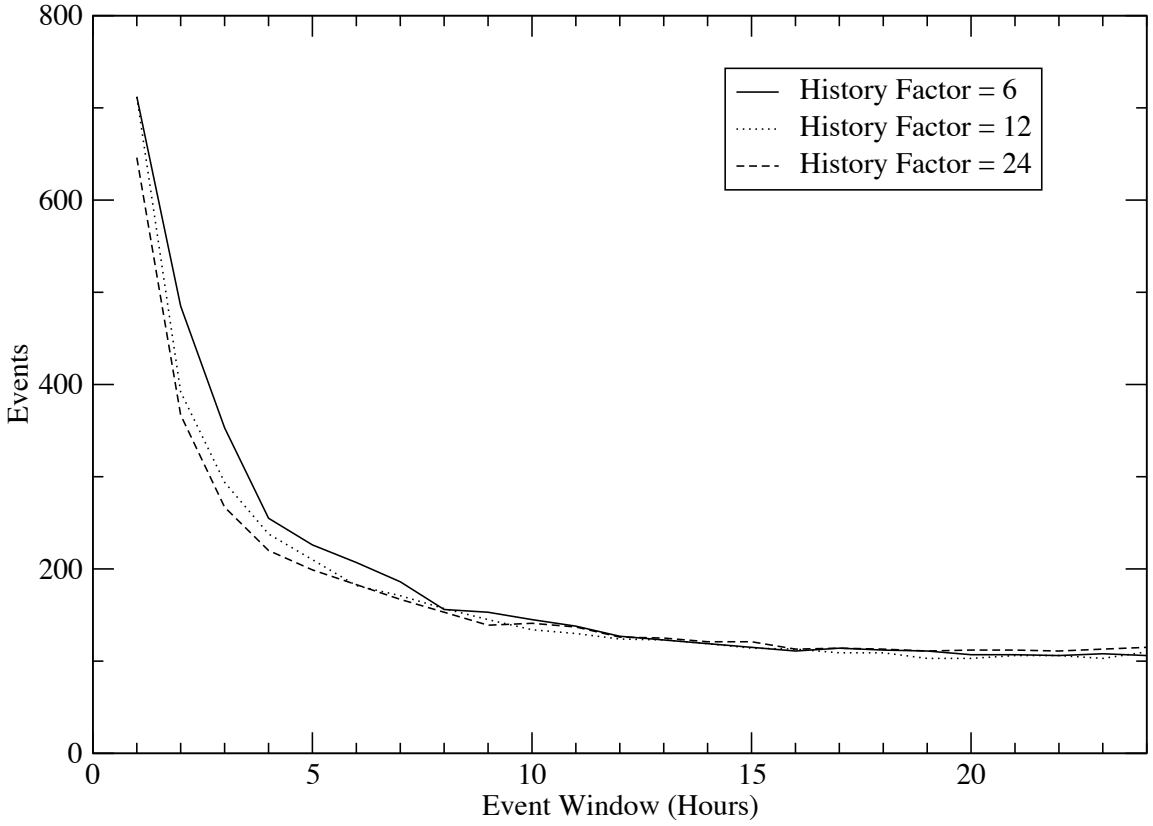


Figure 4.18: The effect of event window size on the number of events generated.

4.3.4.1 Parameterization

As discussed in the previous section, there are four basic parameters that impact the generation of a new event. In this section, we explore the tradeoffs associated with each of these parameters, using data collected at 23 of the 60 IMS blocks from January 1st through April 30th, 2005.

The first parameter we explore is that of the event window. Recall that the event window defines the interval over which the current value is computed via a weighted moving average. Figure 4.18 shows the effect of the event window size, in hours, on the number of events generated for a fixed window size and event threshold. The curve shows that the number of events vary from over 700 to a nearly steady value of 100, with a significant reduction by a value of five hours. One possible explanation for this reduction is that many of the events are in fact short bursts, and longer event window sizes reduce the impact of single-hour bursts.

History factor defines the period of time over which normal behavior is calculated via a weighted moving average. The effect of various history factor values on the number of events is explored in

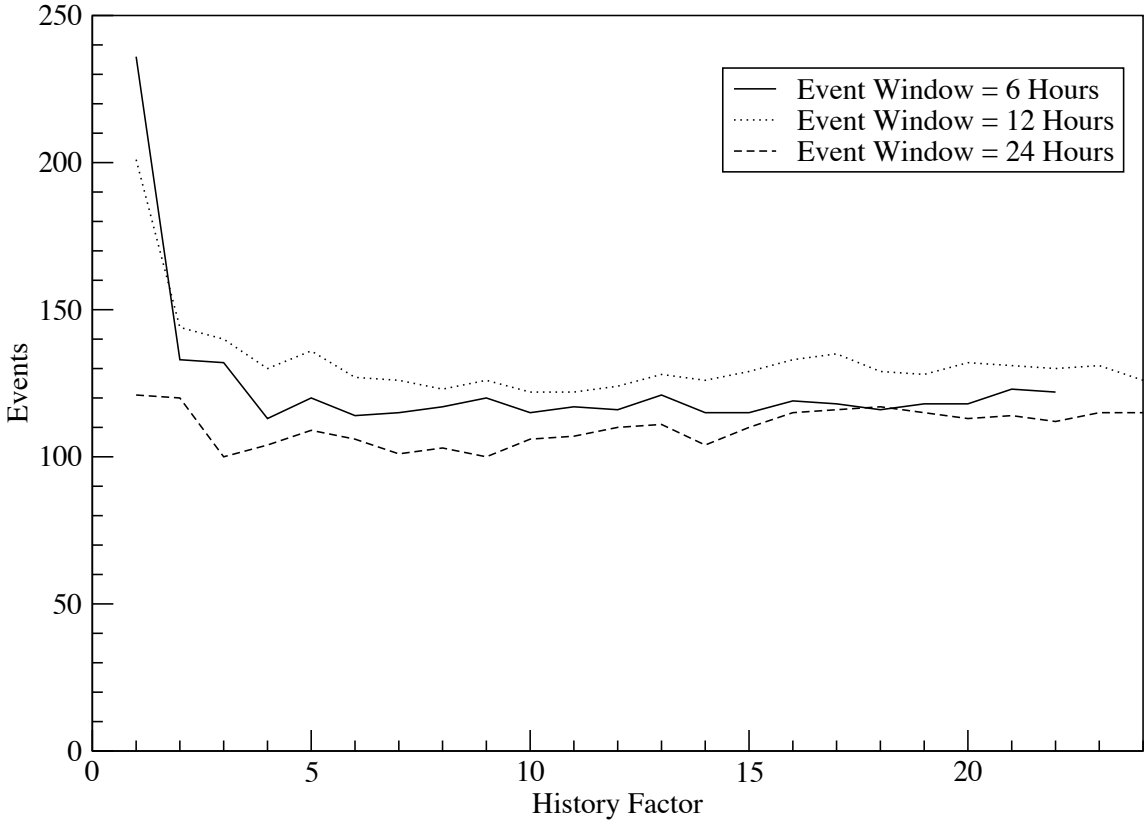


Figure 4.19: The effect of history factor on the number of events generated.

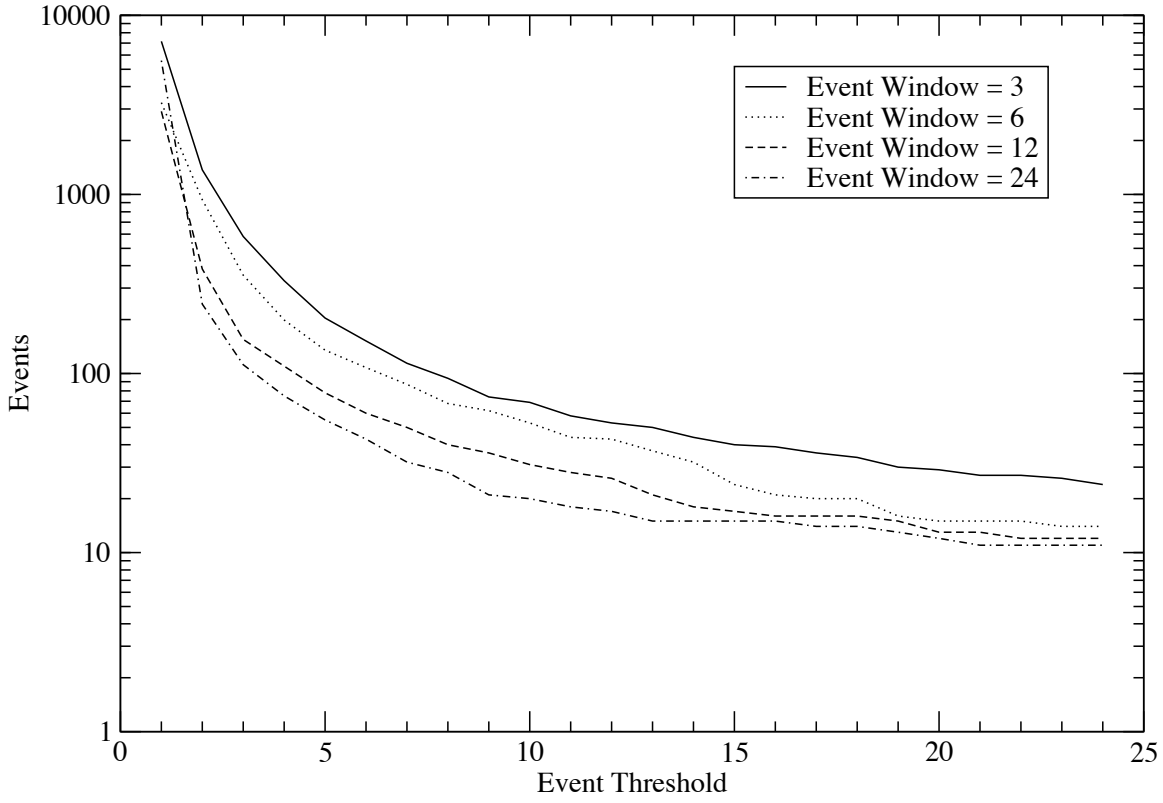


Figure 4.20: The effect of event threshold on the number of events generated.

Figure 4.19. A history factor of two appears to greatly reduce the number of events. It is also interesting to note the crossover in event window size, with an event window of 12 hours creating *more* events than a smaller window of 6 hours. We find no significant difference in protocol or source distribution between these two sets. We are continuing to investigate two additional hypotheses: that sequential scanning activities need more time to reach additional darknets, and that there are frequency components of darknet traffic that are different than that of other Internet traffic.

Figure 4.20 shows the effect of the event threshold on the number of events. The event threshold indicates the degree to which the current value is different than the historic value. This parameter shows the largest impact on events of any of the parameters, with a parameter of one generating an event for nearly every event window (event when the ratio of current to past is one). Drastic reductions in the number of events are seen by event thresholds of three to four.

Coverage represents the percentage of the total darknets monitored that would have individually reported an increase via our algorithm. Figure 4.21 shows the effect of various coverage values as a filter on the number of events. The majority of events generated by the algorithm are seen at a

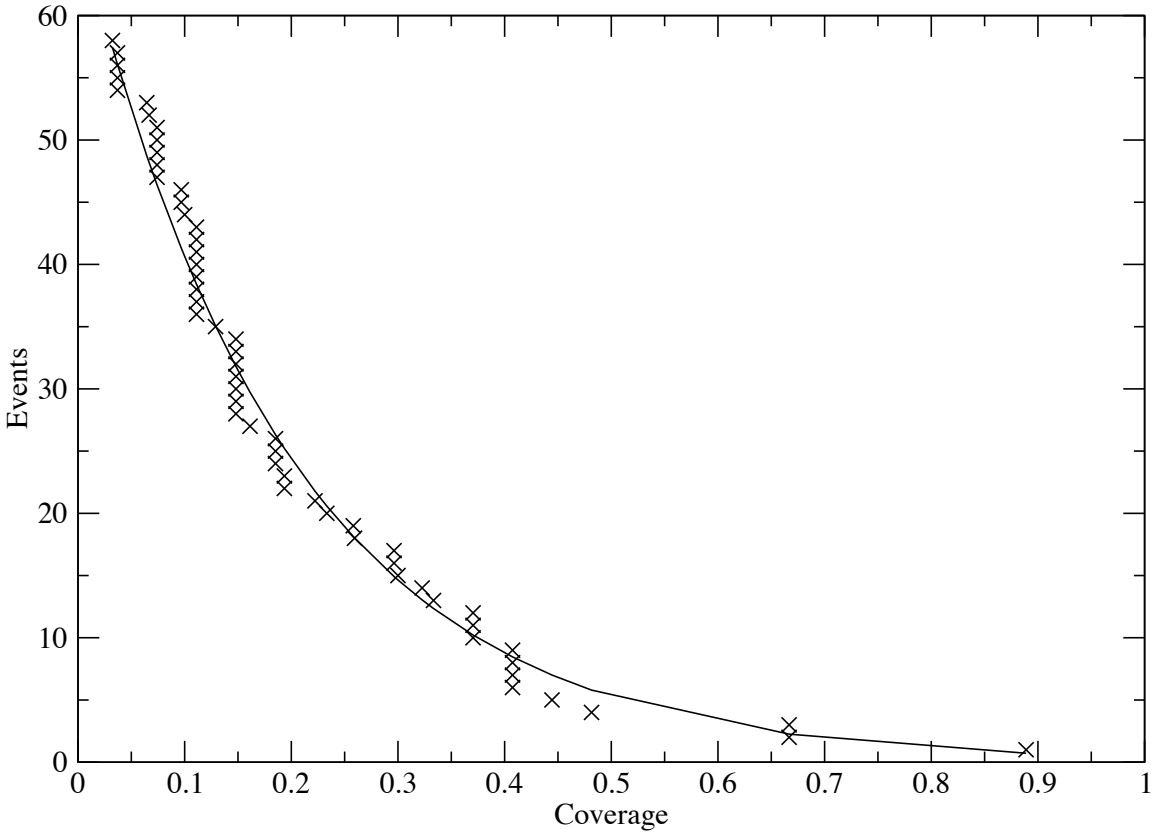


Figure 4.21: The effect of coverage on the number of alerts generated.

Description	Port	Date	Multiple	Coverage
WINS	tcp42	01/13/05 17:31	5.36	0.4815
	tcp42	01/14/05 05:31	61.85	0.8889
	tcp42	01/14/05 17:31	9.77	0.6667
Squid and	tcp3128	02/05/05 11:31	7.73	0.4074
Alt-HTTP	tcp3128	02/05/05 23:31	18.19	0.4074
SYN Scan	tcp8080	02/05/05 10:51	7.53	0.4074
	tcp8080	02/05/05 22:51	20.95	0.3704
MYSQL	tcp3306	01/26/05 09:31	43.89	0.3704
	tcp3306	01/26/05 21:31	8.2	0.4444
	tcp3306	01/27/05 09:31	5.7	0.4074
Syn Scan	tcp5000	01/08/05 14:31	3.42	0.6667
Veritas	tcp6101	02/23/05 21:32	3.54	0.3704
	tcp6101	02/24/05 09:32	3.81	0.3333

Table 4.2: The interesting features identified by the filtering algorithm since January of 2005.

very small handful of darknets. The majority of reduction in the number of events occurs when only considering events that are seen across more than 1/3 of the darknets.

4.3.4.2 Production validation

To help validate our event generation algorithm, we compared the events identified by our system with those identified by the broader security community. It is important to highlight that our system has been in production use over the past few months and has identified several critical new threats. It is used by major network operators and governments around the world to quickly help identify new security events on the Internet.

Over an observation period of four months, the IMS system identified 13 unique events. Table 4.2 shows these events grouped by port. Included in these events are the TCP/42 WINS scanning, the TCP/6101 Veritas Backup Exec agent scanning, and the MySQL worm/bot on TCP/3306. These events demonstrate the quick onset, short duration, and global prevalence of many attacks. Figure 4.22 shows the normalized number of unique source addresses detected at the 23 IMS sensors for these events over time. As the operational test was underway before the completed parametization evaluation, the alerting algorithm used the following non-optimal parameters: alert window of 12 hours, history window of six days, and an alert threshold of three.

Beginning in December 2004, the IMS system observed a significant increase in activity on TCP port 42. This increased activity was notable because Microsoft had recently announced a new security vulnerability with the Windows Internet Name Service (WINS) server component of its Windows Server operating systems. The activity also followed a vulnerability report from Immunity Security on November 24, 2004 that described a remotely exploitable overflow in the WINS server component of Microsoft Windows. Payloads captured on TCP port 42 during the event included sequences that were byte-for-byte identical to exploit code found in the wild following the vulnerability announcement. Although the attack was very broadly scoped, it involved a small number of hosts and only lasted a day. Because the IMS was broadly deployed, it was able to quickly identify this small-scale event in time to enable additional monitoring capabilities.

Another quick attack occurred on January 11, 2005, when IMS observed a substantial increase in TCP port 6101 scanning. Approximately 600 sources were identified as aggressively probing many of the distributed IMS sensors. TCP port 6101 is used by the Veritas Backup Exec agent, a network service that allows for remote system backup. On December 16, 2004, iDefense announced a buffer overflow that enabled remote system-level access. Then, on January 11, exploit code was published for the vulnerability and on the same day, IMS observed a large increase in activity on TCP/6101. Payloads captured from the attack included sequences that were byte-for-byte identical to the exploit code. Once again we see a very quick onset, suggesting the need for an automated system able to rapidly react to new threat activity.

Finally, in late January of 2005, IMS detected a worm/bot targeted at the MySQL database server [65]. The worm/bot propagated by randomly scanning for the Windows version of MySQL on TCP port 3306. The threat was unique because it required interaction with a hidden IRC server before a new instance would start to propagate. The IRC channel was discovered by operators who shut it down, effectively eradicating the worm after a period of a few days. This event shows the importance of having more detailed information on a threat. Without observing the interaction of the worm/bot with the IRC server, there would be no way to discover the communication channel and thus find the simple way to stop the worm.

These three events help substantiate the detection approach and illustrate the value of automated distributed forensics. To further confirm these results and those listed in Table 4.2, we searched for correlations between the port, payloads, and time period of the events with reports from other security data sources. With the exception of the Alt-web and Squid SYN scans, each of these events has been actively discussed in various security forums and corroborated by other monitoring

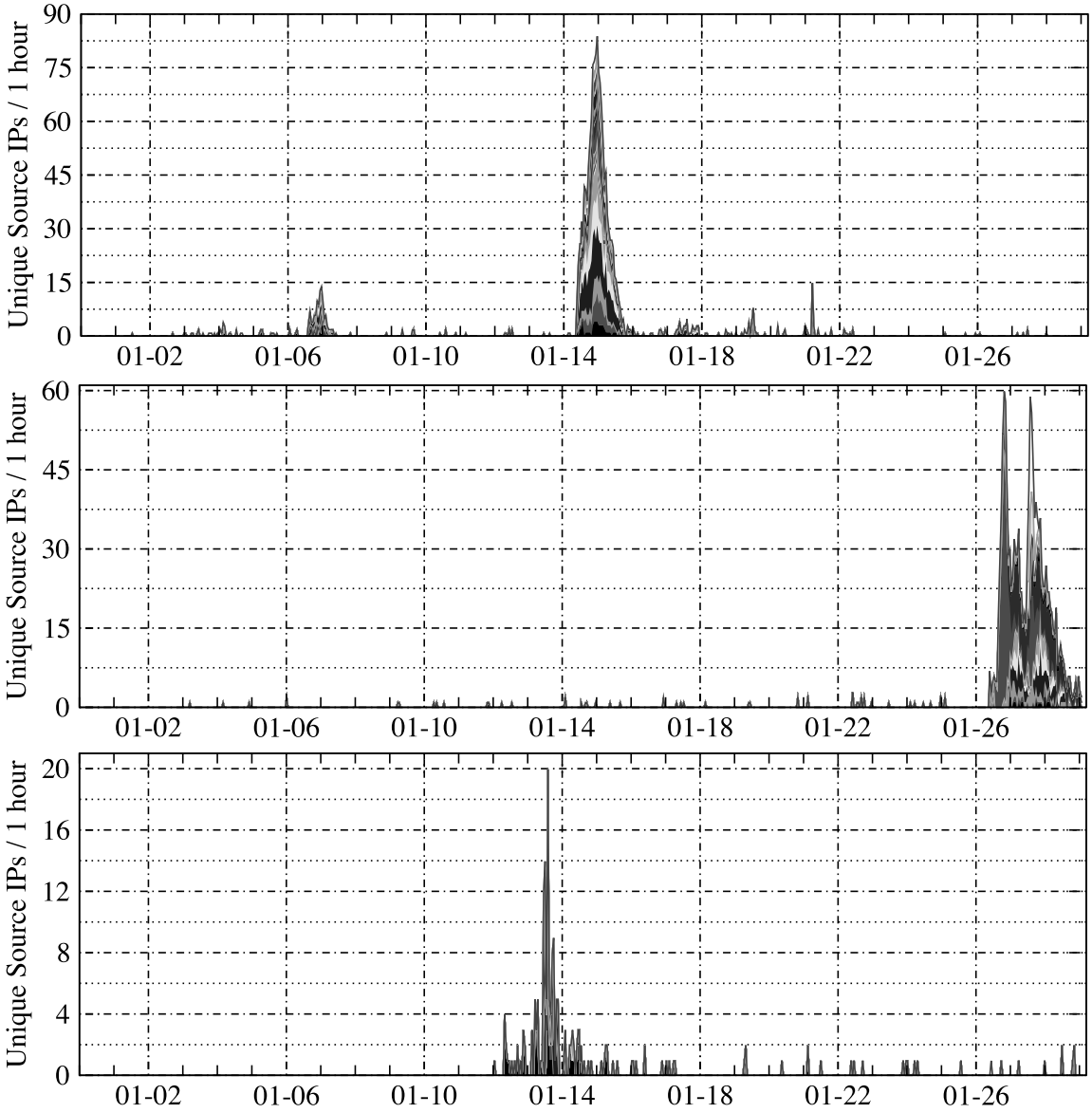


Figure 4.22: The unique source IP addresses over time across 23 darknets to various TCP destination ports for widely publicized events (from top to bottom TCP/42-WINS, TCP/3306-MYSQL, TCP/6101-VERITAS).

projects [117, 18]. There is also evidence that the event identification algorithm did not miss any important events. Analysis of the NANOG, ISN, BUGTRAQ, and FULL-DISCLOSURE emails, as well as the operator logs from ISC and CERT, do not show any major detectable events that were missed by the IMS system. In summary, there is strong evidence that the system was able to identify all the major events during the 4-month period. More time is clearly needed to fully validate the system, but the current data suggests the approach has excellent filtering qualities.

4.4 The goal of filtering

The goal of the filtering algorithms presented in this chapter is to manage cost in the hybrid system while still maintaining both breadth and depth. More specifically we can define the goal of any filtering algorithm in this framework to have the following four properties:

- **Timeliness**. Each new event must be evaluated in the timeliest manner possible. An idealized mechanism would send the first occurrence of any event for analysis.
- **Scalability**. The handoff decision-making algorithm must not adversely affect the ability of the network sensors to monitor large numbers of unused addresses.
- **Accuracy**. The filtering algorithm should not miss any of the events of interest. Ideally, each unique event must be evaluated at least once.
- **Efficiency**. The algorithm must seek to minimize the number of connections sent to the host-based sensors. In an idealized mechanism, each event is evaluated only once.

Each of the two prevalence based algorithm presented here, content prevalence, and source prevalence, make different tradeoffs in attempting to achieve each of these goals. In particular, we have shown in this chapter how the content prevalence algorithm achieves timeliness, scalability, and efficiency. The host-based sensors immediately evaluate each new content payload received, the payload information is elicited without the need for per connection state, and each payload is evaluated only once across the entire network. The major limitation of this technique is its inability to provide strong guarantees on accuracy. Our lightweight content prevalence mechanism suffers two potential drawbacks in this regard: an inability to differentiate threats which have the same first payload, and an over differentiation of threats payloads which may vary by only a few bytes, but are in fact the same threat.

The source-based prevalence mechanism makes an entirely different set of tradeoffs with respect to these goals. In particular, we have shown the source-based methods to be effective at scalability, accuracy, and efficiency. In our evaluation of this technique in early 2005 we demonstrated how the system scaled to a large number of distributed sensors, captured all the human detected events of interest, and reduced the huge volume of data observed to a small handful of events over the 3 month period. The major limitation of this technique is its timeliness. As we show in our evaluation of the various parameters to this algorithm, the current window size plays a large role in the number of events generated, with values less than an hour producing a large number of false positives. While this hour delay is acceptable for some forensic analysis, it is insufficiently timely for quarantine type applications.

4.5 Summary

To provide scalable, early warning and analysis of new Internet threats like worms or automated attacks, we propose a globally-distributed, hybrid monitoring architecture that can capture and analyze new vulnerabilities and exploits as they occur. To achieve this, our architecture increases the exposure of high-interaction host sensors to these threats by employing low-interaction network sensors as front-end content filters. To reduce the load on the host sensors, we filter using two methods: content prevalence and source prevalence. We show that a great number of the payloads seen at an individual sensor are, in fact, duplicates of previously seen packets. We show that, at least for topologically close sensors, these payloads are also visible at other sensors. We show that the emergence of new payloads is an effective way of filtering interactions and that three /24 network sensors generate a mere fraction of a connection per second of load using this method.

In this chapter, we also examined the properties of individual and distributed darknets to determine the effectiveness of building filters based on source prevalence. We showed that individual darknets are dominated by a small number of sources repeating the same actions. This enables source-based techniques to be effective at reducing the number of connections to be evaluated by over 90%. We demonstrated that the dominance of locally targeted attack behavior and the limited life of random scanning hosts result in few of these sources being repeated across darknets. To achieve reductions beyond simple source-based approaches, we look to source-distribution based methods and expand them to include notions of local and global behavior. We saw that this approach is effective at reducing the number of events, by deploying it in 30 production networks

during early 2005.

CHAPTER 5

HOST MONITORING IN THE HYBRID ARCHITECTURE

Effective characterization of Internet threats in the hybrid architecture is dependent on host-based sensors to detect new, interesting events and provided detailed signatures of these event behaviors. In this chapter, we discuss our work on designing and evaluating the host-based sensors, or *honeyfarms*, as part of our hybrid system. In particular, we investigate two very significant problems: what operating systems and applications (i.e., what *services*) to run on the honeynet, and how to detect new events and automate the process of gathering forensic evidence for the host sensors. We begin with a brief survey of work related to honeypots. We then discuss the core issues in host configuration and propose a set of design goals for any honeynet configuration system. We then examine the idea of automating forensics at the host sensors and show how information flow tracking at the operating system provides the necessary detail to characterize threats, but can be filtered to be sufficiently scalable.

5.1 Related Work

The rapid growth in malicious Internet activity due to the rise of threats like automated worms, viruses, and recent bots like AgoBot, has driven the development of tools designed to protect live network and host resources. The fundamental challenge is that the very process of instrumenting the machines and routers in a network can cause significant performance and availability concerns. A detection approach that alleviates many of the availability and reliability concerns of instrumenting live (i.e., production) resources is to monitor non-productive hosts and services–also called a honeypot. Honeypots are network and host resources that do not have production value, and so any interaction with a honeypot is suspicious. An example honeypot deployment might be to configure

a honeypot mail server identical to a production mail server and place it nearby to detect intrusions against the real mail server. Because no clients are configured to use the honeypot mail server, any traffic is the result of an outside process, like malicious activity or misconfiguration. The result is a detection system that requires no modification of kernel, userland, or network resources.

Earlier honeypots were used to lure attackers, understand their tools, and waste their time, so that the actual systems remain unaffected. KFSensor [2] and Specter [5] lure attackers by emulating vulnerabilities. NetFacade [4] emulates a network of machines with vulnerable services so that hackers waste their time and effort. LaBrea Tarpit [66] wastes hackers' time by slowing down the connections. However, all these systems are passive in nature and do not actively intercept traffic for the vulnerable looking honeypots. NetBait [3] and Bait-N-Switch honeypots [1] actively intercept connections and proxy them to a honeypot, if the connection seems malicious. These methods are useful to track hackers who exploit the well-known vulnerabilities. However, many systems are patched for known vulnerabilities, and an archaic configured vulnerability may seem to be a trap, which hackers can identify beforehand. Moreover, attack on archaic vulnerability may not represent the attack on the network hosts, which may contain unknown or newly-announced, unpatched vulnerabilities [121].

Deploying actual physical machines is resource intensive and requires a constant monitoring effort when it gets infected. Therefore, Vrable *et. al.* [119] leverage and speedup virtual machine technology to run multiple operating systems on a single machine with better control of worm infection. However, they still need to determine the virtual machines that need to be configured for large unused addresses.

Many approaches aim to automatically generate worm signatures by observing traffic to unused addresses, and use the signatures to protect other hosts on the network. For example, Honeycomb [59] generates worm signatures by observing traffic destined to unused addresses and Vigilante [40] generates worm signatures by running instrumented machines on unused addresses. However, for any such defense mechanisms to be effective, they need to run the right versions of operating system and services in the right proportions.

Instead of running actual services, the services can be emulated to recover attack details. Service emulation is more scalable and provides easier management. Moore *et. al.* [77] analyzed the visibility of passive blackhole sensors into network attacks for different sizes and locality of the sensors. However, a passive monitor can perform only a coarse characterization in terms of Denial of Service (DoS) attacks [78] and worm propagation on the Internet. Bailey *et. al.* [14] proposed a

lightweight TCP responder (to respond with a TCP SYN-ACK packet for every TCP SYN packet received) to recover the first payload from TCP worms. This simple emulation allowed them to scale to addresses as large as /8, and they were able to recover exploit payload from the Code-Red [36] worm and the Blaster worm affecting TCP port 135. Yegneswaran *et. al.* [126] proposed application level responders to recover exploit details on large unused address blocks. They built custom and stateless service emulation for receiving exploits from known worms. However, partial service emulation to recover payload from known worms may not help them to receive exploit from zero-day worms. Furthermore, the stateless behavior and significant incompleteness in service emulation can lead to anomalous behavior, causing honeynet to be fingerprinted.

Honeyd [84] provides a scalable approach to monitor large unused address blocks by emulating network stack of different hosts on a single machine. Since different TCP implementations return different replies to the active tests of *nmap* [46], a virtual TCP implementation emulating a particular operating system should have replies consistent with the actual TCP implementation. Therefore, the results of different tests on TCP implementations are grouped together under different categories and annotated with the names of operating systems. This annotation allows a security professional to configure unused addresses with desired operating systems, and it helps in collecting exploits affecting a wide variety of systems. To remove inconsistency resulting from incorrect versions of protocol layers on the same machine, Honeyd categorizes possible ICMP and TCP implementations. However, to configure honeyd on an unused address space, one needs to determine the operating systems, services, and their correct proportion on the honeynet.

5.2 Host Configuration

There are several tools that help define and automate the process of creating hosts and services. Honeyd allows the creation of individually consistent hosts by categorizing various possible configurations together into human-understandable personalities [84]. For example, choosing a Windows XP box would automatically configure TCP ports 135, 139, and 445 with appropriate services. This consistency provides some defense against attackers that attempt to probe and identify honeynets by looking for unusual service combinations on a host. For example, IIS web-server on a Linux box is an unusual combination and could indicate the presence of a honeypot. However, these configurations often configure honeynets, which are individually consistent, but provide very narrow views of the services and operating systems on a network. These inconsistencies create a real and

pressing danger to the usefulness of these deployments in that they may not represent the *present* possible attacks on a network. Even worse, not having a representative configuration of the network may make them easy to be fingerprinted, and thus avoided by attackers, in which case the honeynet would miss *future* attacks on the network.

5.2.1 Understand the need for configuration

To represent threats to a network, an idealized approach would create a duplicate network full of honeypots, one for each vulnerable machine. However, with limited addresses and resources to monitor, any honeynet deployment is by necessity an approximation of the live network. In order to aid in the deployment of these approximations, existing approaches to subnet configuration (e.g., Honeyd [84]) break hosts into popular protocol implementations, and to ensure individual host consistency, allow security professionals to categorize various services and operating systems into meaningful personalities. For example, deploying a Windows XP host would automatically configure a host with appropriate services on TCP ports 135, 137, 139, and 445. However, achieving individual host consistency for large diverse systems on the Internet is cumbersome because software is:

- **Large.** Software is increasingly moving from a standalone system to a networked environment, creating large number of protocols for interaction. Moreover, a particular network protocol may not be well specified, creating a large number of different implementations.
- **Complex.** Only a few software packages coexist because of difficult interoperability, default configurations, and the existence of interoperable applications for different platforms.
- **Dynamic.** The frequent release of software upgrades for increased functionality, various security options that network administrators apply differently, and configurable options that users may or may not tune, create an ever changing group of host personalities.

In order to understand the impact of the large, complex, and dynamic nature of configurations on the Internet, we studied the configurations at several large institutions. Table 5.1 shows the type of networks and the diversity in protocol layers (alone and in combination) that we used for our study. A/16 and B/16 were two university networks under our administrative control. To examine enterprise networks, we chose to profile web-server farms that have a significant number of publicly available services. We profiled web-server farms C#1/17 and C#2/19 under one administrative

Network/Mask	Type of organization	Hosts	Webservers	TCP	HTTP	TCP+Ports	TCP+HTTP	TCP+Ports+HTTP
A/16	diverse university network	5512	1237	352	241	1210	699	1386
B/16	diverse university network	1289	169	156	73	392	249	463
C#1/17	web-server farm	11342	10080	256	862	1625	1764	3394
C#2/19	web-server farm	2438	2208	93	293	394	559	811
D#1/19	web-server farm	1859	1513	118	221	330	451	590
D#2/19	web-server farm	1652	1171	137	208	266	417	487

Table 5.1: The number of different categories formed by the combination of implementation and configuration variables.

control and D#1/19 and D#/19 under a different administrative control.

We actively probed to profile these networks. We did this by using a tool to profile TCP, a tool to determine open TCP ports, and an HTTP profiler. We examined only lightweight implementation and configuration tests for these layers. For profiling TCP layer of the hosts, we used different variables associated with nmap [46] tests. However, we removed the test that computes Greatest Common Divisor (GCD) of TCP sequence numbers and sequential increments because of their non-deterministic values. For investigating the services possibly running on the hosts, we examined whether TCP ports numbered from 1 to 1024 are "open" or "closed." For examining the HTTP protocol implementations, we developed a custom code consisting of 38 different tests. These tests examined the parsing and argument checking of HTTP requests, as specified in RFC 1945 [19] and RFC 2616 [43], the methods permitted on a host, the format of header fields returned, support for different HTTP options, and messages for different error replies.

To see the difficulties in configuration for a honeyfarm, consider the data presented in Table 5.1, which shows the large number of TCP implementations, combination of services, HTTP implementations, and the complex associations between them in various networks. In network A/16 we see that the 5,512 hosts result in at least 352 unique TCP stack implementations. Of those 5,512 hosts, 1,237 were running web services, and when probed, yielded 241 differing servers and configurations. Therefore, large, complex, and dynamic behavior of protocol implementations make creating and maintaining consistency using manual methods difficult and time consuming.

5.2.2 Impact of host configuration

Even if all possible configuration and implementations could be categorized together to ensure individual host consistency, this does not ensure that building a network of individually consistent hosts could be created. This is because the categories would be created without observing relationships of hosts within the individual networks. Hence, it would be difficult to assign the correct proportion of each category in a honeynet so that the honeynet proportionately represents the network for all possible combinations of configuration values.

However, this does not mean that the combination of configuration values are not important to be proportionately represented on a honeynet. For a configured host to actually receive an exploit, it must have a configuration representing a vulnerable machine on the network. For example, port 80 should be open to receive the CodeRed exploit, and one has to configure the vulnerable IIS server to allow an infection to take place. Similarly, to receive payload from worms attacking TCP ports

139 and 445, the honeypot should have a responder to establish a NetBIOS session and the actual vulnerable software to allow infection. Therefore, to accurately measure the threat on the network and to provide useful defense as soon as possible, the number of hosts on the honeynet capable of receiving a particular level of threat detail should be proportional to the number of hosts on the network that may receive the same details. Since we do not know before-hand the combination of values that determines the level of details for an attack, any combination of values on the honeynet should be proportional to the network being protected.

Therefore, for any automated configuration system to be successful, we claim it must maintain two key properties:

- **Proportional representation.** The possible vulnerable population on a honeynet should proportionately represent the vulnerable hosts on the network.
- **Individual host consistency.** To be effective in capturing threats and in warding off possible fingerprinting efforts, we would like configuration for each host on the honeynet to be individually consistent.

5.3 Automated Forensics Via Operating System Information Flow

Even with the filtering techniques described in chapter 4 we expect that the host-based sensors will receive a great number of connections to be evaluated. Depending on the data collected at the host sensor, these may then generate a great deal of forensic information to be processed or stored. Therefore, a critical component of any monitoring system will involve an effective measure of automatic detection and forensics evaluation on the host-based sensors.

In this section, we discuss the role of operating system information flow in characterizing threat behaviors. We discuss a variety of heuristic techniques for detection and reducing the information flow in honeypots.

5.3.1 Operating system information flow for forensics

In this system, we chose to track information flow at the operating system level abstraction. At the operating system level, information flows between processes and files. In contrast, a fine-grained instruction level information flow would shows flow between variables or memory locations. In a coarser-grained system, such as that proposed in [42], information is reduced to a set of events (e.g.,

buffer overflows, network writes, disk writes) and information flow is the temporal sequence of events. The advantages of tracking information at the OS level of abstraction are: (1) it does not require access to the program source code or executables, (2) the coarser granularity of information flow at this level requires much lower overhead to track, and (3) the OS-level abstractions allow for an accurate description of the behavior of a system, establishing causality between two events.

The main objects in an operating system are processes, files, and directories. Transient state is kept in the address space of processes. Persistent state is kept in file data and directory data (directories map file names to files). As the system executes, OS-level objects affect each other through various types of interprocess communication (IPC). For example, information flows from a process to a file when a process writes a file, and information flows from a file to a process when a process reads a file. Other ways for information to flow between OS-level objects include process creation, signaling, shared memory, and file name lookups.

Previously others have applied OS-level information flow tracking to the field of computer forensics with the BackTracker system [58]. BackTracker successfully deduced the cause-and-effect chains that preceded an intrusion detection point. Subsequent research has produced the ability to analyze the flow of information forward in time as well as backward in time. The combination of this tool and BackTracker allows us to start from a single detection point, backtrack to allow an administrator to identify the source of an intrusion, and then forward track to identify other objects that have been affected by an intrusion. Figure 5.2 shows an example of an information flow graph produced by BackTracker for the W32-Blaster-Worm-E. This graph shows how information flow can effectively highlight the cause-and-effect chains that led to an intrusion detection point. Figure 5.3 shows the forward tracking graph rooted at the vulnerable process. This graph shows how information flow can effectively highlight the changes that resulted from the intrusion.

5.3.2 Information flow reduction

The main challenge of tracking information flow at the operating system level for hybrid honeypots systems is the number of events to be evaluated in order to detect an interesting occurrence and, once detected, the number of events to be analyzed or presented to the user. Unlike higher level abstractions of events, such as writes to disk or buffer overflows, monitoring events at the operating system level generates a large number of events, even on a honeypot. It is no longer the case that any event is an abnormal one. For example, consider the events in the isolated Windows 2000 Professional honeypot shown in Figure 5.1. This honeypot experienced over 30,000 events in less than

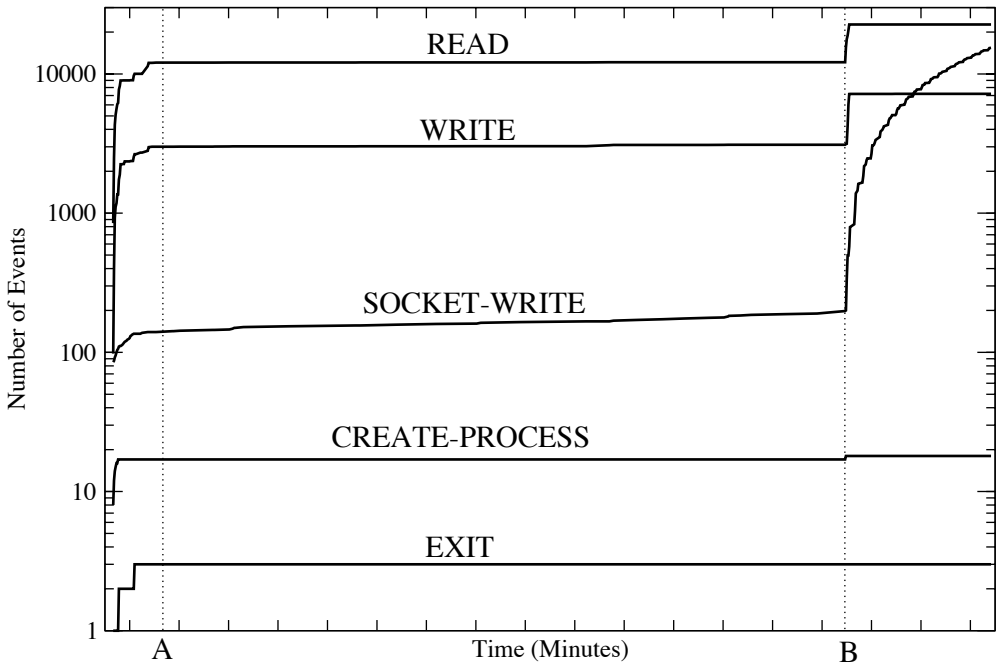


Figure 5.1: The number of events and type of honeypot events over time.

a 20-minute period. Our hybrid system, along with several others, makes use of a proxy or router to send traffic back to the various honeypots in a honeyfarm. To achieve additional scale, these proxies often make use of the fact that a single monitored address is often idle, and send traffic destined to multiple destinations to a single honeypot. It is likely then, that a honeypot in our hybrid system would see a significantly larger number of connections than the idle one presented here.

In order to cope with large numbers of host-based events, we need to develop a means for reducing the information flow. In previous work on BackTracker, the system designers have examined a variety of methods for reducing the amount of information flow data without losing potential vital information about causality (e.g., filtering read only files, collapsing local helper processes). While these techniques offer two orders of magnitude improvements in event reduction, they still leave too many events to be analyzed, especially considering the number of honeypots in a typical honeyfarm. As a result, we need to consider heuristic techniques which may jepordize breaking dependency chains, but offer even greater reductions. Some of these methods include:

- **Filtering by event type**. One very common approach to the display and analysis of host-based information flow is to focus on events of only one type. These approaches generally examine events in one of three types: file events, process events, and network events. Process and network events are particularly attractive given the few number of events normally seen (see Figure 5.1).

- **Filtering by service**. Often what is of interest to an observer is a specific vulnerability or a specific service. Dependency graphs rooted at services of interest (e.g., lsass.exe, svchost.exe) greatly reduce the number of events to consider, but may miss complex interactions.

- **Network dependency graphs**. Any intrusion on a honeypot, since the system lacks an interactive user, is brought into the system via a network connection. Forward tracking dependencies from inbound network connections and backward tracking intuitions, especially subsequent outbound network connections, eliminates those events and objects that are not dependent on any network service and are not able to participate in a network propagating threat.

- **Anomaly detection**. Enumerating all the "allowed" or "normal" activities on a system can provide a basis for both the detection of interesting events as well as the reduction of events that need to be analyzed once something interesting occurs. It is this approach that we discuss in the following section.

5.3.3 Anomaly detection

As one can easily see in Figure 5.1, the vast majority of the events occur in the first few moments of honeypot operation. Enumerating all the events that occur during this boot-up phase and detecting those events that are not part of the initial set of events is one approach to anomaly detection.

It is this temporal anomaly detection that we have made use of in our initial system. While we have been able to successfully use it to deploy and capture information on a wide variety of threats, this approach has several drawbacks that have lead to excessive false positives in the early deployment stages:

- **Scheduled or periodic events**. Modern operating systems include a wide variety of processes that perform system maintenance that operate at a specific time of day or after a specific duration. Examples include disk defragmentation, system registration, or auto update features.

- **First time triggered behaviors**. Some system behaviors are normal, but are only triggered the first time an event occurs. For example, outbound mail or Internet connections may trigger Internet or email connection wizards, or inbound connections to IIS for the first time will result in dllhost.exe being invoked.

- **Lack of configuration sharing across honeypots**. Several of the registry values, files, and other items touched during boot-up will be independent of each other (e.g., Netbios name, MAC address, IP address). These variants will either need to be exhaustively identified, or the profiling will need to be done on each individual machine.

- **Variety of configurations**. Any reasonable honeyfarm will consist of a wide variety of honeypot configurations. Monolithic honeyfarms are neither representative of the target infection population, nor are individual hosts configured with numerous vulnerable applications (e.g.m ISS, MSDE) likely to be representative of real hosts. More configurations mean more profiles.

When properly configured, anomaly detection provides an effective means of determining the events of interest on a honeypot system. However, this approach is not guaranteed to capture all the interesting events, nor show how the various events interact. In order to create this additional level of detail, we apply the following heuristic approach. First, identify all the events created by the anomaly detection system over a short period of time. Run BackTracker on each of these events to generate the causal chain that resulted in the detected activity. Combine each of the backtracker

graphs to identify the common ancestors of the detected activities. Select the nearest common ancestor of all the detection points, that also has an inbound network connection. Use the identified node as the root of a forward tracking graphs, which shows all the activities rooted at that node.

Figures 5.2 and 5.3 show the effect of this algorithm on a Windows 2000 honeypot infected with the CodeRedII worm. The files root.exe, explorer.exe, and numerous outbound connections are the detection points identified by the anomaly detection algorithm. Their nearest common ancestor (with an inbound network connection) is the inetinfo process. A forward tracking graph from this process shows all the relevant activities of the CodeRedII worm.

5.3.4 Application of information flow to the analysis of the persistently infected population

In the previous sections, we have examined the architecture of our hybrid system and discussed operating system information flow as a means for characterizing the relevant activities on a honeypot. In this section, we look at our experiences in using this technique to investigate the properties of the persistent worm population on the Internet.

Over the past few years, the Internet has suffered a number of highly virulent and widespread worms. Today, we find that many of these old threats still scan the Internet in large numbers. It is impossible to point exactly why these threats persist, but the absence of patching combined with lack of expertise and even apathy means that a large vulnerable population still exists [88]. It is easy to dismiss these infected systems as inconsequential; they are not in my network, we have signatures for these worms, the worms don't do anything destructive, etc. However, the true implication of these persistently infected systems is much darker and frightening. At the heart of many DDoS, spam, and phishing attacks is a large pool of compromised hosts sitting in homes and businesses around the world, which provide an anonymous proxy for the real individuals behind these attacks. Persistently infected hosts indicate a large pool of unprotected systems that can easily be (or already are) part of a large pool of compromised systems called *a zombie army* or *botnet*. This trend represents a fundamental shift from attacks against the availability of networks to attacks against individual users.

In the following sections, we investigate the application of our anomaly detection techniques to each of the major worms in the persistently infected population.

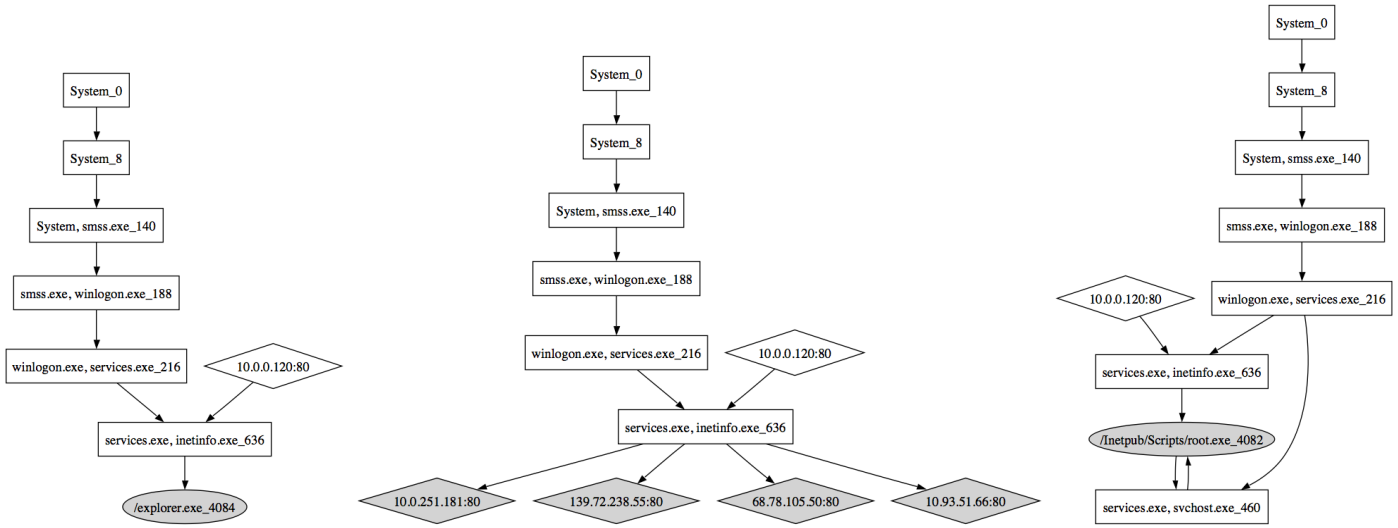


Figure 5.2: Three backtracker graphs, one for each of the three detected events, for a win2k honeypot infected with CodeRedII.

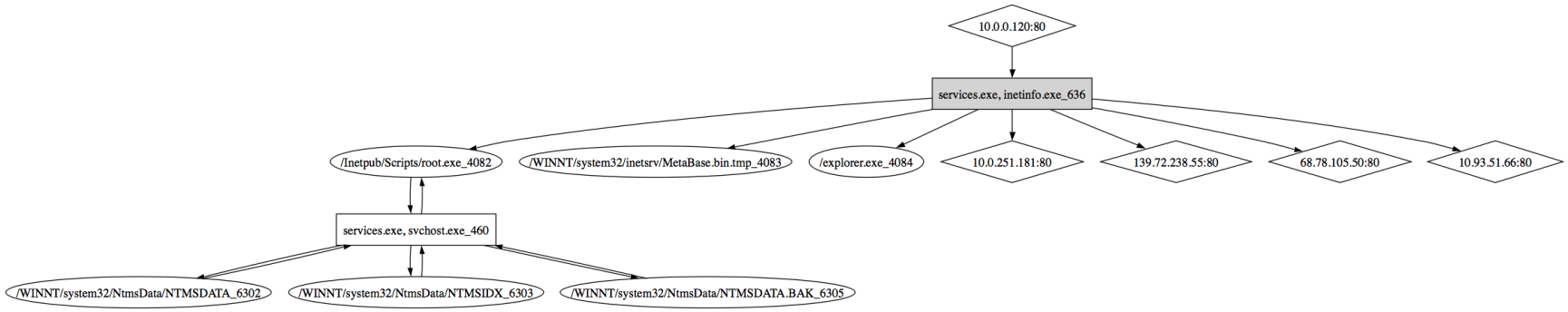


Figure 5.3: A forward tracking information flow graph rooted at the nearest common ancestor of the three backtracker graphs of Figure 5.2.

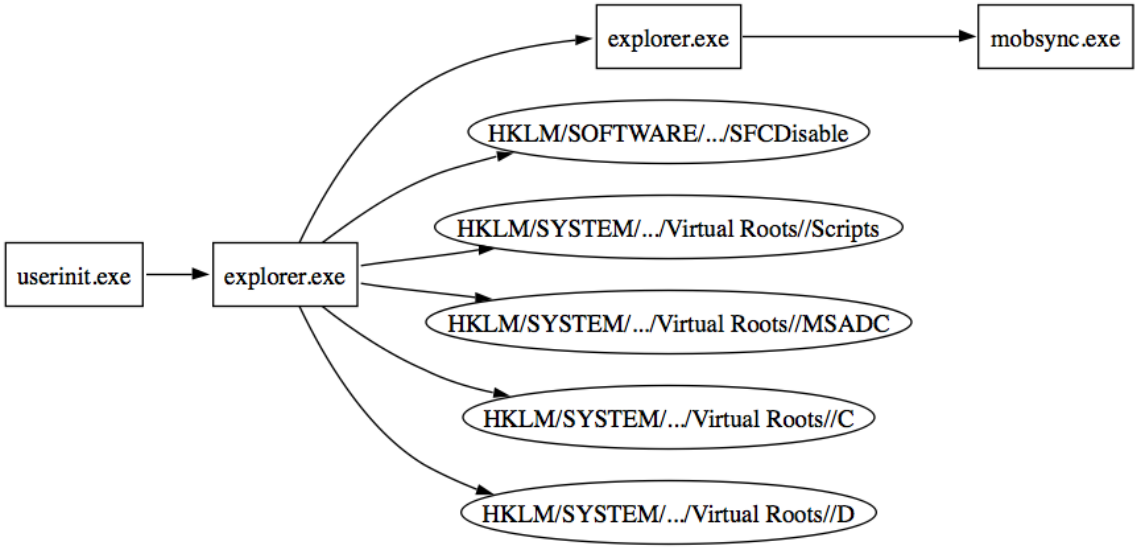


Figure 5.4: The dependency graph for CodeRed after reboot.

5.3.4.1 CodeRed

On August 4, 2001, a highly-virulent new worm, based on a vulnerability used in CodeRed and CodeRedv2 [101] was detected. While exploiting the same vulnerability in Microsoft IIS Web servers, this worm, dubbed CodeRedII, had a completely different payload from the previous two Code-Red's. First, the worm installed a backdoor on the infected system, allowing administrator access to the affected computer, even after a reboot. Second, CodeRedII added *local preference* to the scanning algorithm. The result was that the worm spent more time trying to infect systems in nearby address space.

Figure 5.3 shows the information flow graph for the CodeRedII worm, including its outbound scanning and placement of a copy of the cmd.exe file as root.exe in the scripts directory of the Web sever, essentially allowing command line access to the machine over the Web. One particularly interesting facet of the worm is the placement of a Trojan-ed explorer.exe in the base windows directory. The filename, explorer.exe, is that of a legitimate service used to run programs when a user is logged in, but this is typically run from the system directory. The effect of this Trojan-ed file can be seen in Figure 5.4, which shows the same honeypot after reboot. Due to a bug in PATH creation, the explorer in the base directory is run instead of the correct one. This version adds virtual directories to the Web server path, essentially opening up the entire hard drive to access from the Web. The program then terminates and calls the real explorer to perform normal operation.

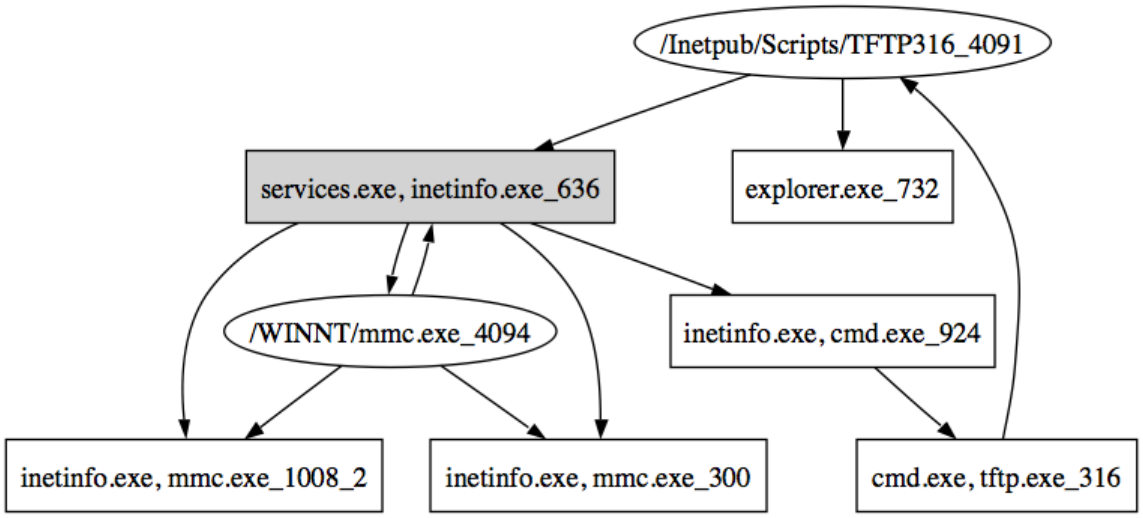


Figure 5.5: The information flow graph for Nimda.

While the write to the explorer.exe file was visible in the initial intrusion, it wasn't until after an externally triggered event, in this case a reboot, that the behavior of this Trojan-ed file was obvious. These types of worm payload activations, whether through user interaction or remote interaction, are particularly difficult to capture in an automated fashion.

5.3.4.2 Nimda

The Nimda worm emerged in mid September, 2001, and it was notable in that it was one of the first publicly successful multi-attack vector worms. The worm, which initially appeared to some as a variant of the CodeRedII worm, could spread through a wide variety of ways. The worm was capable of spreading using the same vulnerability in IIS servers used by CodeRedII. In addition, it was a parasitic virus, capable of exploiting the backdoors previously created by CodeRedII infected machines. For example, if scanning for IIS vulnerabilities a susceptible Web server might show:

```
GET /scripts/root.exe?/c+dir HTTP/1.0
GET /c/winnt/system32/cmd.exe?/c+dir HTTP/1.0
GET /scripts/..%255c../winnt/system32/cmd.exe?
  /c+tftp%20-i%2010.0.0.120%20GET%20Admin.dll
  %20d:\\Admin.dll HTTP/1.0
```

The worm spread through email attachments, making use of a bug that automatically ran .exe files. It used emails found on the local host to send infected email out to the network. It scanned for open network shares and placed trojan'ed DLL files, which would execute the next time the user opened them. On infected servers, the worm modified Web pages to include code that asked Web browser clients to download the infected payload.

The information flow graph for one instantiation of Nimda is shown in Figure 5.5. One of the more interesting aspects of this worm is that it changes its behavior based on how it entered the system. In the case of Figure 5.5, we see the graph that resulted from a server infection, and the execution as mmc.exe, and the placement of Admin.dll as the payload. If the virus is run in client mode, either by email attachment, Web page, or through an infected file, a completely different behavior is observed. This dynamic worm behavior illustrates one of the more difficult aspects of automating this detection, as not only might we require different configurations to see different worm behaviors, but that worms may need to enter the system in different ways.

5.3.4.3 Blaster

On Wednesday, July 16, 2003, Microsoft Security Bulletin MS03-026 [37] announced a buffer overrun in the Windows RPC interface that could allow attackers to execute arbitrary code. The flaw, which was initially uncovered by the Polish security group LSD [81], affected many Microsoft Windows operating system versions, including NT 4.0, 2000, and XP. Infection by the Blaster worm occurs on TCP port 135. If a connection to TCP port 135 is successful, the worm sends an RPC bind command and an RPC request command containing the buffer overflow and exploit code. The exploit opens a back-door port on TCP port 4444, which waits for further commands. The infecting system then issues a command to the newly infected system to TFTP the worm binary on UDP port 69 from the infecting system and to execute it.

Figure 5.6 shows the information flow graph for two different variants of the Blaster worm, Blaster-A and Blaster-F. One of the interesting things to note about the information flow graph is the similarity between members of the family beyond the shared point of entry. The threat carrier and automatic activation are the same between all the worm variants, with the only difference being the dropped file. While this seaming bodes well for systems that would like to quickly determine families of threats, it also poses some interesting challenges. While the worm authors left some early profiled differences (e.g., the name of the dropped file), these easily could have be obfuscated (e.g., random name and size for all variants). The real differentiator, the worm payload behavior, is

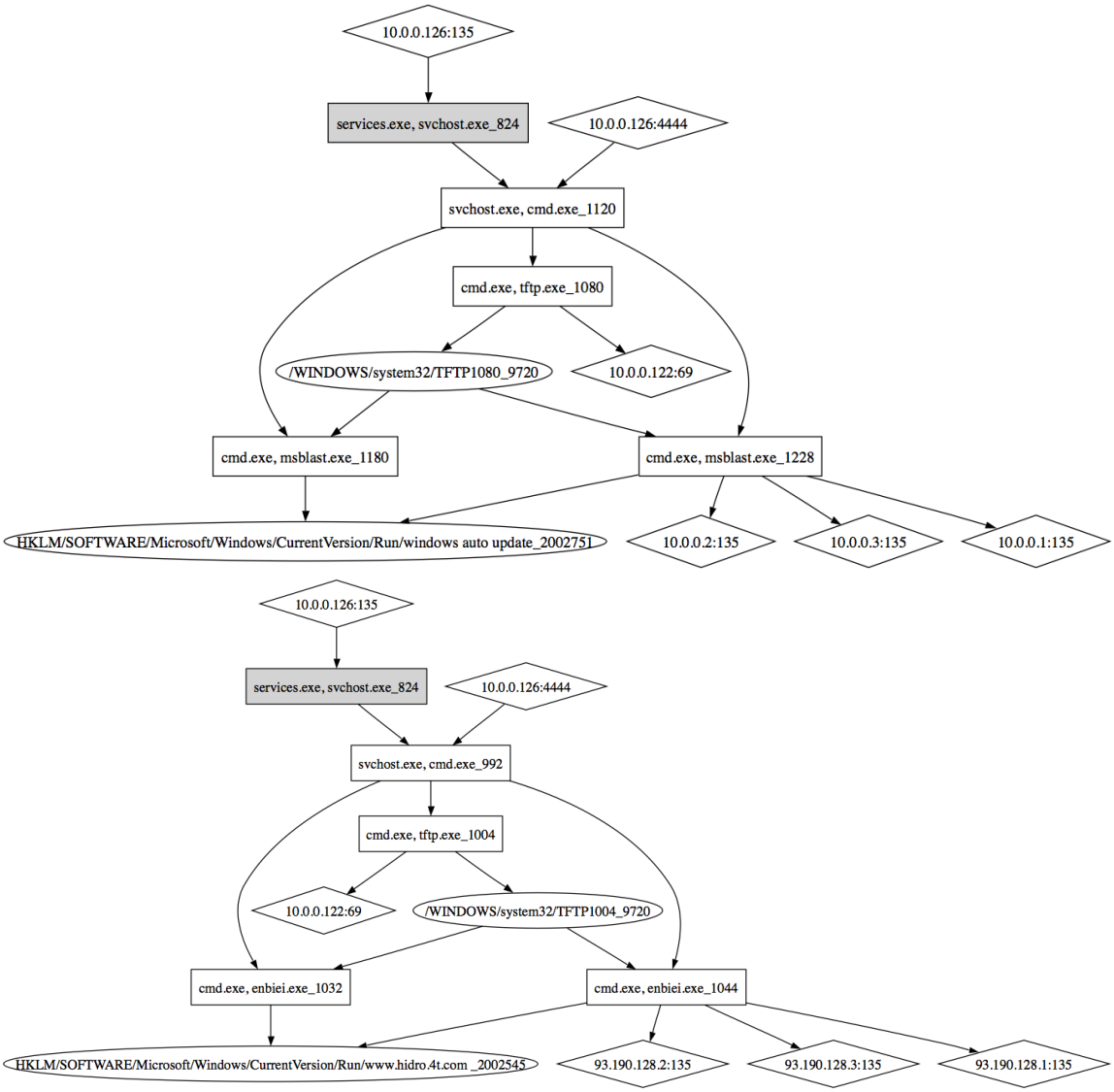


Figure 5.6: The information flow graph, rooted at svchost.exe, for the A and F variants of the Blaster worm on a Windows XP honeypot.

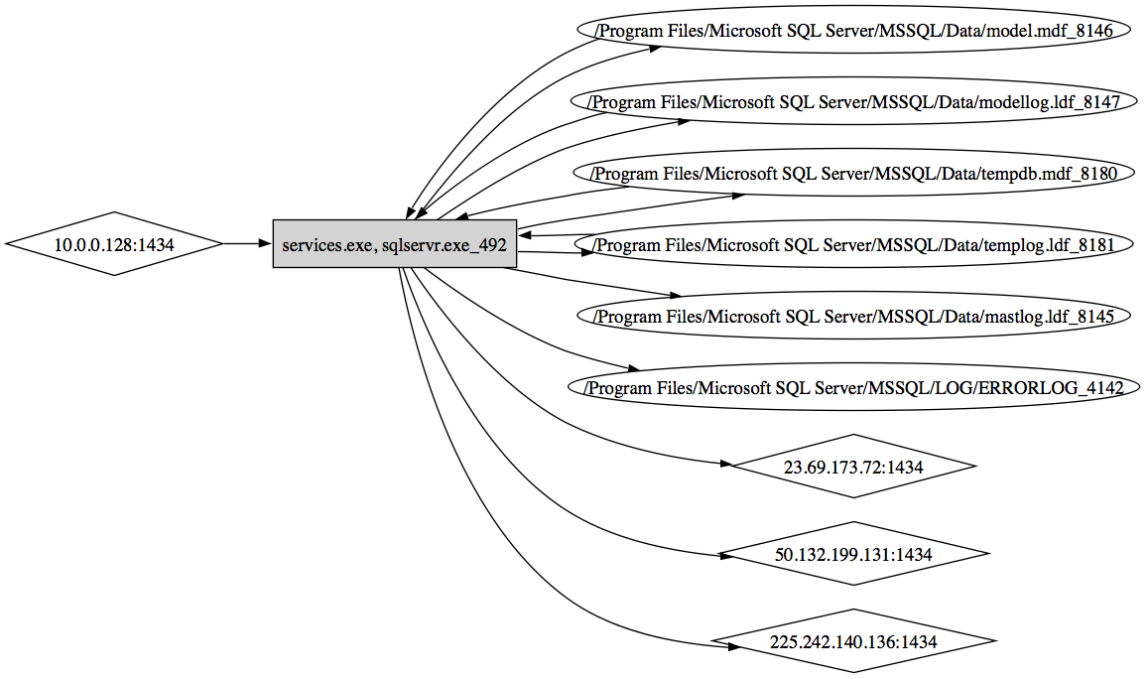


Figure 5.7: The information flow graph, rooted at sqlservr.exe, for a Windows 2000 honeypot infected with the Slammer worm.

actually not visible. This is because Blaster had a fixed window of time in which its payload was instructed to launch attacks hard coded into the worm.

5.3.4.4 Slammer

The Slammer worm, also known as Sapphire, infected more than 90 percent of the more than 75,000 vulnerable hosts within 10 minutes [76]. A side effect of the rapidly propagating Slammer worm was a huge amount of worm scan traffic causing wide-spread network congestion and even serious outages. The Slammer worm utilized a buffer overflow vulnerability in Microsoft SQL Server 2000 on the SQL Monitor Port (UDP port 1434). The Slammer worm itself is a very simple memory-resident worm with a tiny footprint (376 byte UDP payload). The small size of the worm, combined with the speed of using state-less UDP rather than state-full TCP, meant the Slammer worm could attempt new infections at an incredible rate.

The information flow graph for the Slammer worm, shown in Figure 5.7, shows the relative simplistic behavior that can make up a self-propagating exploit. It also highlight two important points in information flow tracking: the danger in monitoring only one type of event type, and the

broader tradeoffs between virulence and information hiding. As one can see, Slammer is memory resident, does not create any new processes, and does not write to any files. Therefore, without a network detection there would be no information flow graph. Fortunately, Slammer was exceptionally virulent from the network perspective and is quite easy to detect. It does, however, highlight a broader concern about the tradeoff attackers make between detect-ability and speed. Had the authors made Slammer's propagation pattern follow that of a "normal user," say by embedding it in other requests or following normal communication patterns, this threat would not have been detected via information flow analysis.

5.4 Summary

In this chapter, we presented our work in designing and operating the host-based sensors as part of our hybrid architecture. While our work on these host-based sensors is not complete, we have investigated two of the key issues associated with building these host-based sensors: how to configure the sensors and how to automate detection and forensics.

First, we showed that even a small campus network can have a *surprisingly* large number of TCP, port, and application configurations. In order to avoid fingerprinting, but continue to assure the sensors ability to capture the properties of the vulnerable population, we claim that any automatic configuration systems must provide both consistent and representative profiles.

Second, we investigated how to automate forensic gathering on host-based sensors and found it can be achieved through the use of operating system information flow. We describe the need for filtering operating system events, even on honeypots, and offer several methods for event reduction. We investigate the notion of anomaly detection, both as a means for event detection, but also for event reduction and we describe a method for building plausible graphs of host behavior based on this method. We then use these techniques to investigate their effectiveness in characterizing the persistent worm population. We show that, while effective at providing valuable forensic information, the techniques that automate forensics analysis must deal with several worm properties, which make their efforts more difficult including; user- or timing-based activation, internal target list discovery, and embedded propagation mechanisms.

CHAPTER 6

EXPERIENCES CHARACTERIZING, MEASURING, AND TRACKING INTERNET THREATS USING THE HYBRID ARCHITECTURE

In the previous four chapters, we have introduced the hybrid architecture for characterizing, measuring, and tracking Internet threats. We have examined each component of this system in turn, including the network-based sensors, host-based sensors, and the intelligent filtering. While each of these provided their own unique contribution to the field of measurement, the purpose of these efforts was to create an architecture capable of providing detailed characterization of threats. In this chapter, we examine how the hybrid architecture has been used to provide a variety of useful characterizations about Internet threats. These characterization efforts fall into two broad categories:

- **Analysis of individual events**. Most of the efforts in characterizing DarkNet traffic have focused on generating analysis of individual threats. Analyses of these events includes information on the size of the event, the size of the vulnerable population, and the distribution of that population geographically or topologically.

- **Analysis of trends**. While timely analysis of individual events can be informative, combining these analyses over time can lend insight into trends in behavior and capabilities of these new threats. These point analyses are often combined with coarse-grained characteristics of DarkNet traffic, such as the absolute number of packets or bytes seen by a sensor, breakdowns by protocol, port, or the three majors classes of events (backscatter from denial of service attacks, worm or manual scanning, and misconfiguration). In combination, these longterm analyses provide a degree of predictibility and help with proper preparation, which is key to an effective response to any threat.

In this chapter, we examine each of these types of analyses, in kind. We begin with a look at the deployment of the hybrid architecture. We then examine examples of individual threat analysis, including an evaluation of the Blaster worm, the SCO denial of service attacks, and the Bagle backdoor scanning. We next study three key trends: the shift in demographics, threat persistence, and the trend toward more dangerous threats.

6.1 Distributed Deployment

One key contribution of the hybrid system is its ability to support a widely-distributed deployment. The current deployment consists of 60 distinct monitored blocks at 19 physical installations (see Table 6.1). We monitor 17,096,192 IPs, which is roughly 1.15% of routed IPv4 space. These deployments range in size, from a /25 to a /8, and include major service providers, large enterprises, academic networks, and broadband providers. An effort has been made to represent geographic diversity, as we include sensors in Asia, Europe, and the US. These sensors represent a range of organizations and a diverse sample of the routable IPv4 space, including 31 of all routable /8 address ranges (or 21% or routed /8 networks).

The first version of the IMS was deployed in 2001 as part of a global threat monitoring system developed by researchers at Arbor Networks and the University of Michigan. This initial installation was built to monitor a /8 network, or approximately 1/256th of the total IPv4 space. Between 2001 and 2003 this system processed several hundred petabytes of network data, including recording millions of scan and backscatter events. It was able to characterize numerous Internet worms, such as CodeRed, Nimda, Sapphire, and Blaster. In 2003 the system was updated and expanded to represent the distributed architecture presented in the networking section. In late 2005, the host component was developed and added to a single /24 monitored block at the University of Michigan.

The current system revision contains lightweight responders across all ports and a data query engine to support distribution. The query system consists of a query daemon that is installed on each sensor and listens for queries from a portal server. The portal server acts as a query aggregator, forwarding queries to each sensor and forwarding back the responses. The portal server also runs a web application that utilizes the query system to provide real-time views of threat traffic in the IMS sensor network. While the query system is an interesting implementation on top of the IMS architecture, it is beyond the scope of this paper. The filtering components discussed in Chapter 4 are currently utilized to generate detailed analysis of new threats observed by the system.

/26 x 5	/23 x 2	/20 x 8	/17 x 3
/25 x 1	/22 x 4	/19 x 1	/16 x 9
/24 x 18	/21 x 2	/18 x 6	/8 x 1

Table 6.1: IMS Deployments

6.2 Individual Event Observations and Experiences

In this section, we demonstrate the ability of the hybrid system to capture and characterize several important Internet threats that were previously not possible or were difficult using passive monitoring methods. While we do not present all our observations over the last 5 years here, we do provide illustrative examples of individual threat analysis in the following categories:

- **Internet worms**. Many scanning worms seek out new targets to infect by scanning large blocks of address space (or the Inter Internet). The hybrid system has been successful in characterizing threats from a wide variety of these new worms including CodeRed, Nimda, Slammer, Blaster, Witty, Slamer, Bobax, and MySQL.

- **Scanning and Botnets**. The hybrid system has been useful in detecting a wide variety of scanning activities. These include scans for vulnerable systems, such as recent WINS and Veritas scanning, as well as opportunistic scanning for previously infected hosts, such as scans targeting the Bagle-Backdoor.

- **DDoS**. The distributed architecture enables observations of DDOS events, including those that may not generate much traffic or target a wide range of addresses, such as the 2004 denial of service attacks against SCO.

6.2.1 Internet worms

Internet worms represent a class of security threats that seek to execute code on a target machine by exploiting vulnerabilities in the operating system or application software [79, 123]. Unlike viruses, however, worms do not rely on attaching themselves to files to propagate. Rather, worms stand alone and propagate, by using the network to scan for other potentially vulnerable hosts, and then exploit them without user interaction.

Globally-scoped network monitoring systems, such as the IMS, are helpful in characterizing,

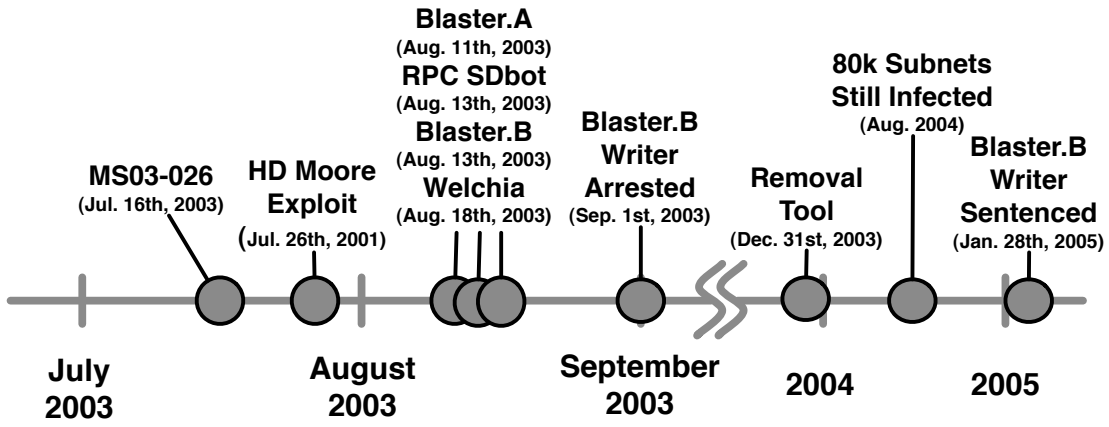


Figure 6.1: A timeline of the selected events from the Blaster worm and related families.

measuring, and tracking these threats. The IMS has been able to provide valuable insight into a variety of worm behaviors, including the following:

- **Worm Virulence**. How much traffic resulted from this worm? What routers or paths were most congested by this worm?
- **Worm Demographics**. How many hosts were infected? Where are these hosts geographically, topologically, and organizationally? What operating system are the infected hosts running? What is their available bandwidth?
- **Worm Propagation**. How does the worm select its next target?
- **Community response**. How quickly was policy employed? Which organizations were affected quickest and who responded quickest? Who is still infected?

As an example of the type of analysis made possible by the hybrid system, consider the following analysis of the Blaster worm. On Wednesday, July 16, 2003, Microsoft Security Bulletin MS03-026 [37] announced a buffer overrun in the Windows RPC interface that could allow attackers to execute arbitrary code. The flaw, which was initially uncovered by the Polish security group LSD [81], affected many Microsoft Windows operating system versions, including NT 4.0, 2000, and XP.

At the time of the vulnerability disclosure, there was no known public exploit and a patch was made available through the Microsoft web site. CERT [25] and other security organizations

followed with their own advisories over the next several days. Almost immediately, discussions of the vulnerability on security lists began. By Saturday, July 26, 2003, HD Moore [48] had published a working exploit "dcom.c" on the Full Disclosure mailing list.

Scattered reports of people reusing the exploit appeared over the next several weeks and then, on Monday August 11th 2003, the first variant of the Blaster worm struck. The Blaster worm, also known as MSBlast or Lovsan, copied code directly from the dcom.c exploit and added its own code to launch a coordinated denial of service attack. The attack attempted to exhaust resources of the *windowsupdate.com* web site using a TCP port 80 SYN flood. The worm also used the exact same backdoor mechanism from the example exploit to transfer the worm payload to newly infected systems.

The decompiled source code from the initial Blaster variant reveals the details of the worm's unique behavior [92]. The Blaster worm can be launched in one of two ways: the worm can be started as the result of a new infection or when an already infected machine is rebooted. Once launched, the worm immediately starts the setup for further propagation. It chooses an address from the same local /16 (Class B) address as the infected host. Next, the worm picks a random number to determine whether to use the local /16 address it just generated or a completely random address. The bias is 60% for a completely random address and 40% for the local /16 address. Next, the offset is chosen to determine if the exploit will infect Windows 2000 or Windows XP. This is also chosen randomly, 80% XP, 20% 2K. Then, on certain system dates, a thread to launch a DoS attack against *windowsupdate.com* is started. There are no further calls to the random number generator. However, throughout this process the random number is repeatedly seeded with the number of milliseconds since boot time. This indicates a significant lack of understanding of random number generators on the part of the worm author. The impact of these poorly seeded random generators is discussed elsewhere [35]. The propagation setup is complete when the worm uses the previously generated starting address and exploit offset to attempt infections to 20 sequential addresses using 20 threads on TCP port 135. This process is repeated on the next 20 sequential addresses, indefinitely scanning sequentially through IPv4 space. If a connection to TCP port 135 is successful, the worm sends an RPC bind command and an RPC request command containing the buffer overflow and exploit code. The exploit opens a backdoor port on TCP port 4444, which waits for further commands. The infecting system then issues a command to the newly infected system to TFTP the worm binary on UDP port 69 from the infecting system and to execute it.

Numerous Blaster variants, as well as several new families of worms that exploit the same initial

RPC vulnerability, have appeared since its release, many of them emerging within a few weeks of Blaster. Perhaps the two most notable uses of this vulnerability are the Welchia [90] and SDBot [89] worms. Welchia or Nachi as it is sometimes called was an *anti-worm* [21] that attempted to patch the vulnerability and ended up causing significant damage of its own. SDBot was notable in that it used the same RPC vulnerability to install the SDBot kit. The SDBot kit creates a backdoor on the system, which enables remote control of the infected system through IRC.

The impact of the Blaster worm was not limited to a short period in August. A recently published survey of "19 research universities showed that each spent an average of $299,579 during a five-week period" to recover from the Blaster worm and its variants [45]. The cost of this cleanup effort has helped solidify a growing view of worms; not as acts of Internet vandalism, but as serious crimes. While the original Blaster.A author is still on the loose, authors of several other variants have been apprehended [83, 93].

6.2.1.1 Observations of Blaster in August of 2003

One of the more interesting elements of the Blaster worm is that it provides an excellent example of a worm life cycle. While all worms do not follow this cycle in its entirety, it is still informative in understanding broad behaviors. A four-phased worm life cycle consists of latency, growth, decay, and persistence. Figure 6.2 shows the four-phased life cycle of the Blaster worm as observed by our measurement infrastructure.

Latency describes the period of time between the discovery of a vulnerability and its observation as a worm in the wild. This period may or may not include the publication of the vulnerability, release of patches, and theoretical or working exploits. Following the release of the initial advisory from the LSD group, along with an advisory and patch from Microsoft, our measurement infrastructure observed a strong upsurge in the number of unique hosts scanning TCP port 135. Prior to this vulnerability disclosure, the infrastructure observed between 4 and 10 unique sources per day for this activity. Following the advisory in mid-July, 2003, this number increased to 100-300 unique scan sources per day. In addition, several higher activity spikes were observed during this period, each correlating to the release of successive exploit tools from various underground individuals or groups.

In the *growth* phase, the worm strikes and begins to infect the vulnerable population. This period is often described using epidemiological models from population growth and biology, and their application to random scanning worms is fairly well understood [108, 101, 127]. During

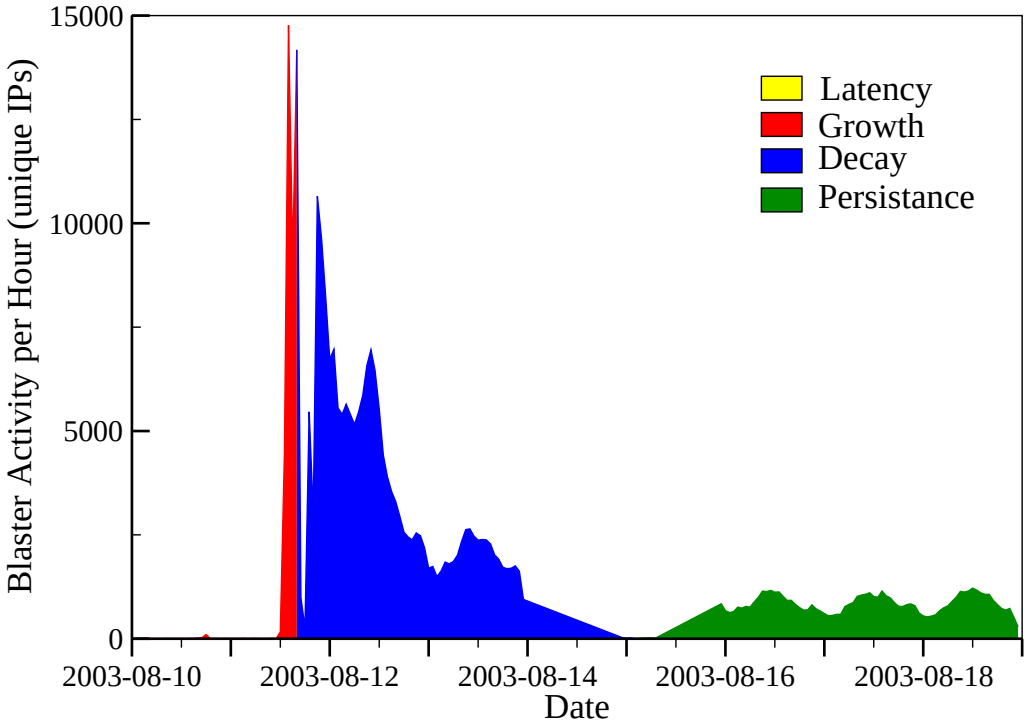


Figure 6.2: A snapshot of Blaster worm, showing the four phases of the worm life cycle.

the first few hours of its growth, the Blaster worm spread exponentially with a doubling time of approximately 9 minutes. However, as the prevalence of the worm increased, this doubling time slowed, and our observations peaked at nearly 15,000 unique IP addresses scanning TCP port 135 in a single, one-hour period, with roughly 106,000 unique sources in a 24-hour period. Reverse DNS lookups for the active hosts in the peak hour of Blaster's activity shows a global distribution of hosts, as shown in Table 6.2. Analysis of the second-level domain names shows that a significant number of consumer broadband providers were affected by the worm's spread.

The growth phase is followed by a *decay* phase in which infected systems are patched, removed from the network, or policy is deployed to minimize their impact. Within 8 hours of the worm's initial outbreak, the number of unique hosts per-hour scanning for TCP port 135 against 256 contiguous addresses began to diminish. Fitting this loss of worm activity to a simple exponential decay, we calculate a half-life of approximately 12 hours. This loss of activity continues for approximately 4 days, through the end of the work week, as shown in Figure 6.2.

Finally, most non-destructive worms enter into a *persistent* phase. This phase is characterized by a relatively smaller population of hosts that remain infected. Following the decay phase, the observed Blaster activity reached a fairly consistent level. Figure 6.3 shows Blaster activity in late August 2003 compared to activity one year later. The initial growth in observations in August 2003 correlates with the appearance of the Welchia worm on August 18th. The last two days represent the steady state seen for the next several months. The activity in both the 2003 and 2004 observations displays a circadian pattern, with peak activity occurring near midday in the eastern United States.

6.2.1.2 The Blaster worm now

The Blaster worm was released well over a year ago, providing ample opportunity for individuals and organizations to clean up infected machines. One might expect the worm to decay quickly with only a handful of hosts still infected today. The reality is that the Blaster worm is not only still scanning the Internet today, but the infected population is larger then expected.

Observations performed during the two months starting in August 2004, revealed over 200,000 unique IP addresses, with peaks of 4,100 unique IP addresses per day, and 500 unique IP addresses per hour. While DHCP has been shown to create overestimates of the population when considering periods longer than a day [101], the slow scanning rate of the Blaster worm and the short daily operational lifetime of a host may lead to under counting when only considering daily or hourly values. In an effort to at least partially account for the DHCP effects, we considered only the first 24

TLD	Aug 2003	Aug 2004
net	39.3%	34.2%
com	15.4%	19.6%
jp	3.8%	7.7%
fr	3.1%	2.1%
ca	1.8%	1.3%
de	1.7%	5.3%
br	1.5%	2.0%
it	1.2%	3.5%
au	1.2%	0.0%
edu	1.1%	0.1%

Table 6.2: TLD analysis of unique source IPs on August 11th, 2003 and a year later.

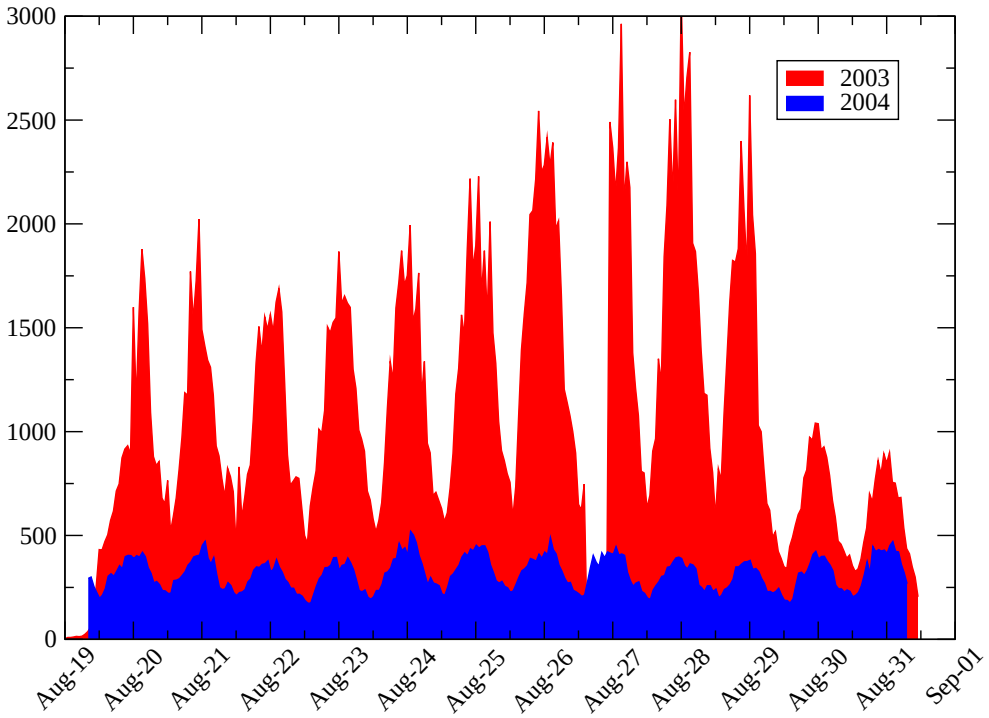


Figure 6.3: Unique Blaster hosts per hour in late August 2003 and over the same period in 2004.

bits of an IP address under the assumption that most DHCP address assign addresses from a small pool of IPs in the local subnet. This analysis showed roughly 90,000 unique addresses during the two-month period, suggesting that the number of infected hosts is much greater than the 4,100 IP address per day lower bound.

The persistent Blaster population also exhibits a very interesting activity pattern that mirrors a circadian cycle, as seen in Figure 6.3. Closer inspection shows that on average, Monday sees the most traffic and Saturday the least. In addition, the peak activity typically occurs from around 10:00 to 15:00 GMT, which corresponds to working hours in the United States. This pattern suggests that the infected systems are power cycled every day as workers and home users turn on and off their computers.

The observation that many old worms persist is not new [104], but the huge number of unique hosts still infected with Blaster today is very surprising. Simply put, these persistent infections are due to people who either don't know or don't care enough that their machine is infected. A surprising data point was the release of the Blaster removal tool by Microsoft on January 6, 2004 [38]. This tool was supposed to remove the worm executable and install the patch needed to prevent re-infection. Although the tool was downloaded by millions of users, Blaster observations for January 2004 changed very little, suggesting this tool had little effect on the larger infected population.

One approach to gaining insight into the persistence of Blaster is to compare the infected population observed during the outbreak with the currently infected population. Looking first at reverse DNS data, we find that the two populations have similar demographics, as is shown in Table 6.2. A rough comparison of the number of US-based infections during the outbreak with its persistent population shows them to both be roughly 55%. Overall, the top-level domain distribution across countries changed very little between outbreak and persistence.

The number of infected systems in different countries appears to have remained relatively stable over the year; however, the actual addresses have changed quite dramatically. 99.5% of persistent addresses are not found in the outbreak population and 99.5% of outbreak addresses are not found the persistent population. It is possible a large number of infected hosts simply moved within the same LAN due to DHCP or other administrative decisions. In an effort to minimize this effect, we compare only the first 24 bits of the host IP address. We found that 73% of the persistent addresses are not found in the outbreak population, and 85% of outbreak addresses are not found the persistent population. Thus, there is very significant movement between subnets. As a final test, we performed the same analysis comparing only the first 16 bits of the address and masking out the last 16 bits.

The results showed that 37% of persistent addresses are not found in the outbreak population, and 21% of outbreak addresses are not found in the persistent population. Hence, there is still substantial churn in the infected networks between the outbreak and infected populations.

Our observations indicate that the Blaster worm does not appear to be going away anytime soon. Recent tools targeted at eradicating the worm appear to have had little effect on the global population. In addition, analysis of the persistent population shows that the infection appears to have successfully transitioned to new hosts as the original systems are cleaned or shut off. The prognosis is not good. This evidence suggests that the Blaster worm will remain a significant Internet threat for many years to come.

6.2.2 Scanning

The second example of analysis enabled by this architecture focuses on scan activity. Scanning of networked computers has been around almost as long as networked computers have existed. Benign types of scans were originally used to verify network connectivity (ICMP) or forwarding paths. As applications and services began to use the network, scans for potential vulnerabilities were developed. Individuals looking for services that were vulnerable to attack scanned networks, exploited those servers, and gained control of the computing resources. With the advent of auto-rooters, and complex scanning tools, probes to networks from other machine on the Internet are now a routine occurrence.

One artifact of more recent worms is that after compromising a system, these worms commonly install backdoors in the system they infect. These backdoors have long been a hypothetical source of personal data, computation, and network resources. The IMS has the ability to respond as if these new services were running, allowing the collection of the scan payload. An interesting application of this data has been in investigating the degree to which hackers have been trying to utilize this potential resource through secondary infections.

Starting on approximately March 20, 2004, IMS began tracking significant amounts of scanning on backdoor ports left by widespread variants of the Bagle [7] and MyDoom [6] mail-based worms. The patterns of sources for this traffic show that they are widespread. The payloads identified suggest that these hosts may be under attack from another worm, such as AgoBot [116], or opportunistic attackers, such as those looking to build armies of compromised computers.

Bagle and MyDoom are families of SMTP-based worms that began spreading in early 2004 and propagated via mass mailer routines. Both of these mail-based worms have many variants that have

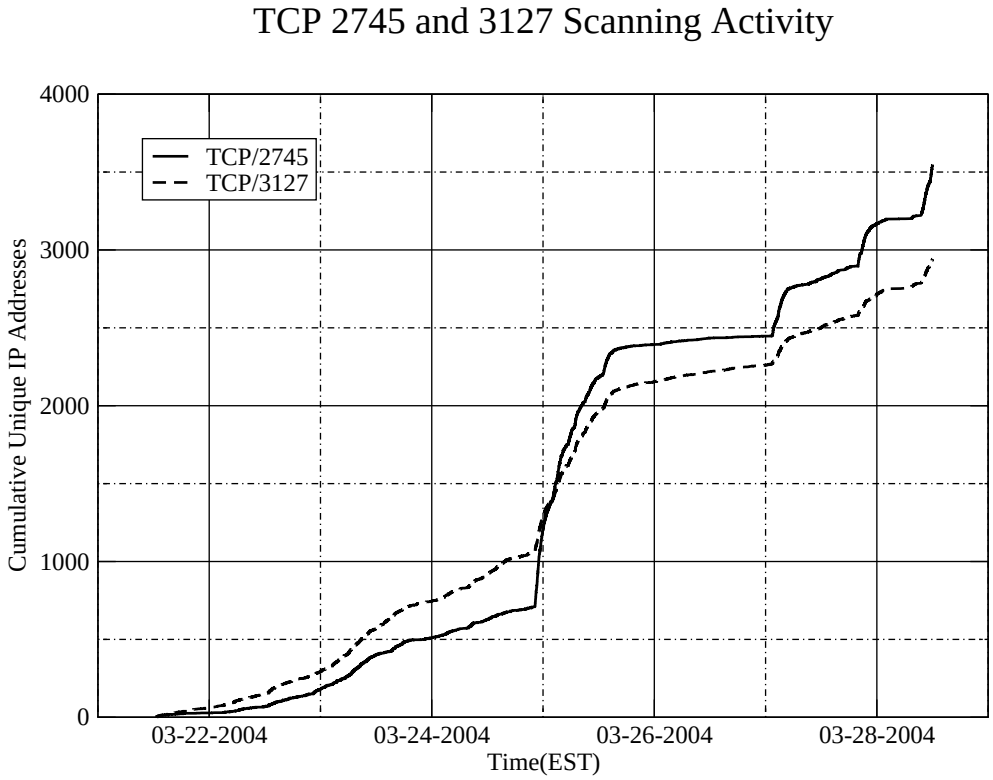


Figure 6.4: The cumulative number of source IP addresses contacting 2745/TCP and 3127/TCP over a one week period as observed by one /24 sensor

rapidly appeared in a short time span, with new variants appearing almost daily. The relationship between these families is interesting and reveals a fight in the virus world between authors. Some of these mail-based worms attempted to uninstall the others, disrupting their spread.

Each of these malware families listen on TCP ports as a backdoor mechanism for remote control of the hosts. In the case of many of the Bagle variants, it is TCP port 2745. In the case of the MyDoom family of mail-based worms, this port is typically TCP port 3127. Access to these ports could be used to upload arbitrary software to an affected host or to execute arbitrary commands.

Figure 6.4 shows the scan traffic for 2745/TCP and 3127/TCP over a one-week period, starting March 20th of 2004, as observed by one /24 IMS Sensor. Scans against other ports opened by the Bagle variants (including 6777 for Bagle.A, 8866 for Bagle.B, and 11117 for Bagle.L) have not been observed in any appreciable quantities. In addition, the IMS has not observed significant

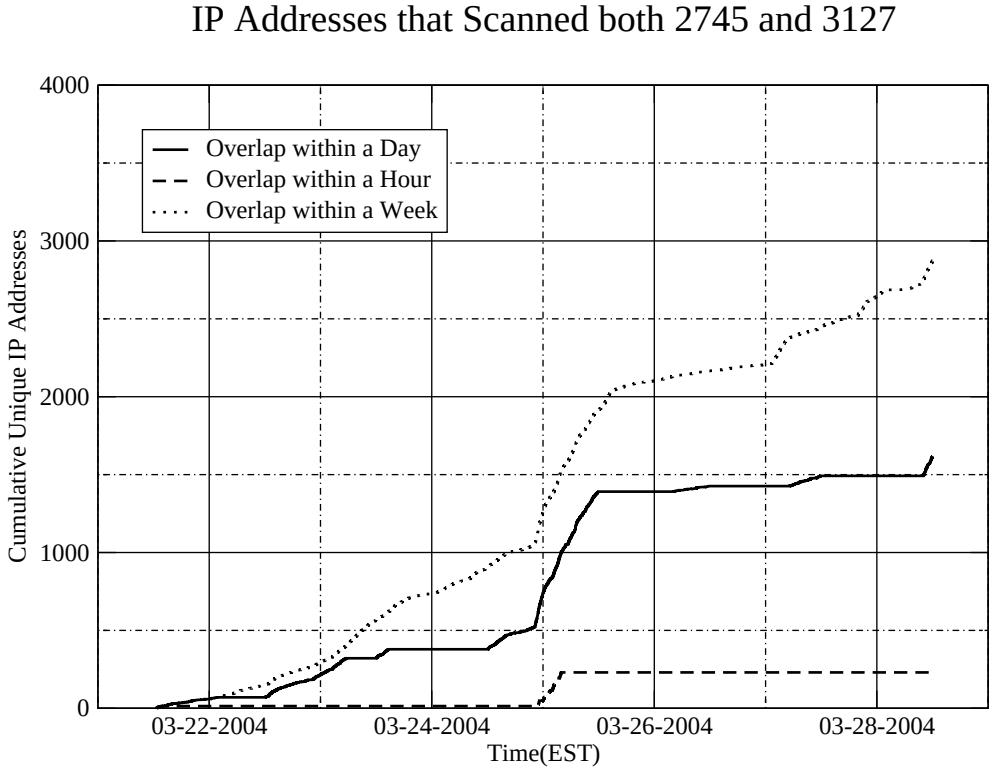


Figure 6.5: IP address overlap between 2745/TCP and 3127/TCP sources.

activity in this time period against additional MyDoom ports, including TCP ports 3128-3198.

While the similar magnitude and shape of the datasets in Figure 6.4 may suggest that these scans are occurring concurrently from the same source addresses, this is actually not the case. Figure 6.5 shows the cumulative unique source addresses that scanned both 2745/TCP and 3127/TCP. The three datasets represent three distinct definitions of overlapping: source addresses that scan both ports within the same hour, within the same day, and within the same week. This view shows us two interesting facts. First, sources that scanned one port did not scan the other port in the same hour, indicating that these were not lock step scans. Second, when observed over the course of a week, nearly all sources scanned both ports, indicating that while the target selections were independent of each other, the same pool of source addresses were being used for both scans.

The top payload captured using IMS on port 2745/TCP (backdoor ports left by the Bagle.C-G and Bagle.J-K mail-based worms) is shown in Figure 6.6. Note that the signature in Figure 6.6

```
43 ff ff ff 30 30 30 01 0a 28 91 a1 2b e6 60 2f 32 8f 60 15 1a 20 1a 00
```

Figure 6.6: Top payload against port 2745/TCP seen in scanning.

differs from the Bagle removal mechanism described by Joe Stewart [110]. The similarity of the first four bytes (0x43 0xff 0xff 0xff) must be noted; after that point the signatures are divergent. This may indicate a common command or authentication byte sequence.

The top payload for the MyDoom worm scans (target port is 3127/TCP) appears to be a UPX packed binary, suggesting new software is being uploaded. The origin and function of this binary is unknown. Both the source and destinations of these scans against TCP port 3127 appear to be widespread. Sources are in many global networks, typically in consumer broadband networks and academic networks.

The activity on the Bagle and MyDoom backdoor ports captured by the IMS clearly demonstrates the value of the lightweight responder. Without the ability to respond as if these new services were running, the IMS would never have been able to collect the payload and analyze this activity. This event also shows the advantage of using a simple lightweight responder over handcrafted service modules. Because the backdoors observed were from relatively new worms having a huge number of variants, it would be extremely time consuming to create services modules for each variant in time to capture these events.

6.2.3 Distributed Denial of Service Attacks

The final event presented here exhibits how IMS has visibility into Denial of Service attacks. Denial of Service attacks seek to deny legitimate users access to resources. Denying service typically takes the form of either crashing a computing resource through some bug in the software implementation or by consuming a finite resource. Distributed Denial of Service attacks (DDoS) are a subset of this class of attacks that rely on a typically large number of end hosts to consume network resources (e.g., host connection queues, available link bandwidth).

On December 10, 2003, shortly after 4PM EST, a long-lived denial of service attack began against a single web server address for The SCO Group (www.sco.com). The two largest components of this attack can be seen in Figure 6.7. Because these attacks utilized spoofed source addresses, the IMS system was able to observe some of the backscatter from the attacks. The de-

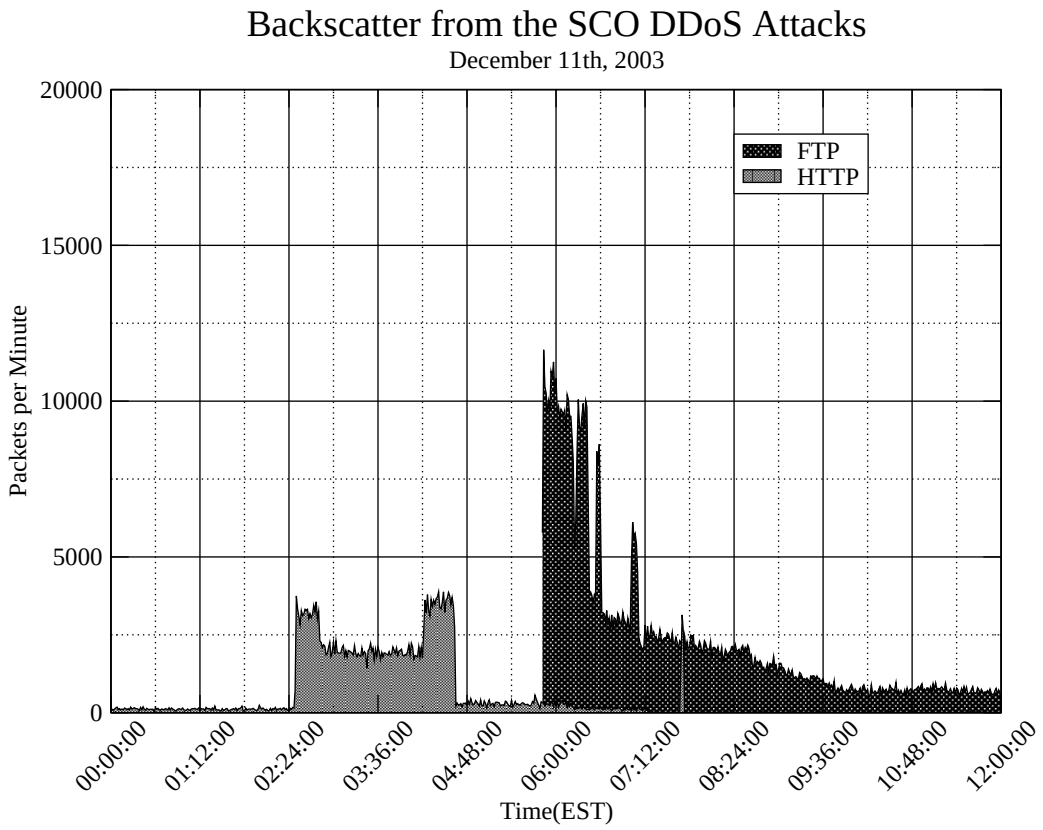


Figure 6.7: The two largest events of the December 2003 SCO DDoS attacks.

Table 6.3: Discrete DDoS events targeting SCO

Start	Duration	Type of attack
Dec. 10, 16:24 EST,	902 minutes	SYN flood port 80, www.sco.com,
Dec. 11, 05:49 EST,	480 minutes	SYN flood port 21, ftp.sco.com,
Dec. 11, 05:51 EST,	16 minutes	SYN flood port 25, mail.ut.caldera.com,
Dec. 11, 07:28 EST,	27 minutes	SYN flood port 80, www.sco.com,
Dec. 12, 12:23 EST,	13 minutes	SYN flood port 80, www.sco.com,

tection system classified the attacks as five discrete events. Three of these events were against the web servers for The SCO Group, one was against their FTP server, and one was against their SMTP server. Additional analysis, such as source host fingerprinting through passive techniques and TTL-based distance measurements of attacking populations, are not presented but represent part of a spectrum of analysis enabled by IMS.

The events depicted in Table 6.3 illustrate that DDoS events can be long lived and made up of several smaller events combined to create larger attacks. Furthermore, this event shows that attacks using spoofed source address are still in use.

Denial of Service attacks represent an interesting demonstration of the utility of the IMS and the need for address diversity. Because attacks may randomize their sources addresses over the entire Internet, smaller swathes of address space may not be able to accurately determine the scope and magnitude of an attack (e.g., a /24 may only see .000005% of the backscatter, while a /8 may see .5%).

6.3 Trend Observations and Experiences

While individual event analysis provides useful information such as forensics for mitigating or filtering an ongoing event, it is when the information from numerous such analysis is combined that the value of such a hybrid system truly emerges. By combing characteristics across events, we are able to lend insight into long term trends in threats and threat behaviors. These types of analysis have a broad impact, influencing the design of new security systems and defining research in the

areas of security, networking, and software engineering. In this section we examine several trends in threat characteristics, whose observations were made possible through the use of the hybrid system.

6.3.1 Demographics

One of the more disturbing threat trends is the increasing scope of the vulnerable population. The total number of machines on the Internet continues to grow very quickly. ISC reported over 285 million Internet hosts in their 2004 survey, over 100 million more than their previous years results [52]. While it is not surprising that the Internet continues to grow quickly, it is certainly troubling that they are liable to be running increasingly vulnerable operating systems and applications. In a recent evaluation of vulnerabilities, CERT reported a nearly two-fold increase in reported vulnerabilities every year from 2000 to the 2004 [26]. In addition, attackers are moving away from the notion that worms must target large vulnerable populations. A prime example is the recent Witty worm [100] that attacked a specific firewall product from a specific vendor.

Perhaps an even more disturbing trend is the increasingly distributed populations residing in different spheres of jurisdiction and administrative control. By some estimates, the US still has 50% of the Internet users. However, a recent survey by the UN showed that the rate of growth in the US is slowing considerably and was dramatically less then the growth in Asia, Europe, and Africa [118] over the past few years. This growth increases the number of jurisdictions and leads to difficulties for organizations trying to mitigate attacks. An example of this is a DDoS attack targeting a company in Europe using computers from many providers in the US, being controlled by an IRC server in China. In addition to the increased globalization, we continue to see rapid growth in the number of autonomous systems [70]. Each of these systems, even within the same legal jurisdictions, bring their own set of expertise, security and traffic policy, and user agreements further complicating communication and cooperation.

As the number of Internet hosts and applications grow, threats have increasingly larger vulnerable targets. This growth also brings larger populations residing in different spheres of jurisdiction and administrative control. The result is a large, fragmented, and vulnerable population that presents an increasing challenge. These trends can be seen through the evolution of various threats as well. Table 6.4 shows the distribution, by TLD, of many of the more recent worms. Note not only the shift away from Asia as the top spot, but also the inclusion of a variety of smaller domains.

TLD	CodeRed	CodeRed2	Nimda	Blaster	Witty	Sasser	Bobax
net	31%	29%	21%	30%	21%	61%	53%
none	37%	37%	60%	36%	29%	14%	19%
com	7%	8%	4%	12%	23%	3%	5%
kr	10%	17%	8%				
edu	4%	2%	1%		9%	5%	
jp				3%	3%	1%	1%
fr	1%	1%		2%		2%	1%
de	1%	2%	1%	1%			2%
cn	1%	3%	2%				
it	1%	1%				1%	2%
nl	1%				2%		1%
br	1%			1%			2%
tw			1%		1%	1%	
ca				1%	2%		
au						2%	
mx							2%
es	1%						
dk	1%						
pl							1%
se					1%		

Table 6.4: The TLD distributions of several popular worms.

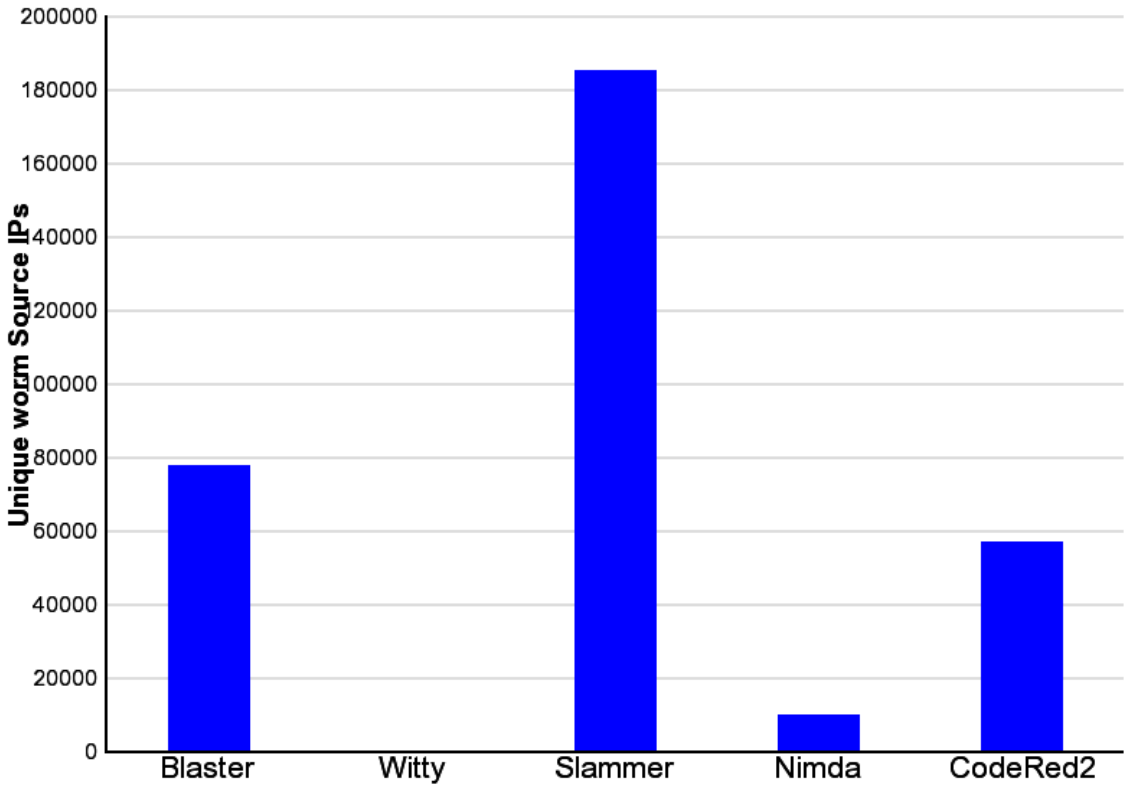


Figure 6.8: The number of unique hosts observed infected with a persistent worm over a one-month period in August 2004.

6.3.2 Persistence

Over the past few years the Internet has suffered a number of highly virulent and widespread worms. Today, we find that many of these old threats still scan the Internet in large numbers. It is impossible to point exactly why these threats persist, but the absence of patching, combined with lack of expertise and even apathy, means that a large vulnerable population still exists [88]. To measure the magnitude of the persistent population, we performed a month-long observation of five significant worms using the IMS in August of 2004. The number of unique IP addresses observed for the Blaster worm (first released Aug, 2003), the Witty worm (March, 2004), the SQL Slammer worm (Jan, 2003), the Nimda worm (Sept, 2001), and the CodeRedII worm (Aug, 2001) are shown in Figure 6.8. The number of infected hosts are staggering, ranging from tens of thousands of hosts for the Nimda worm to hundreds of thousands of hosts for the Slammer worm. Amazingly, worms from over three years ago, like CodeRed, still number in the thousands today.

It is easy to dismiss these infected systems as inconsequential: they are not in my network, we have signatures for these worms, the worms don't do anything destructive, etc. However, the true implication of these persistently infected systems is much darker and frightening. At the heart of many DDoS, spam, and phishing attacks is a large pool of compromised hosts sitting in homes and businesses around the world, which provide an anonymous proxy for the real individuals behind these attacks. Persistently infected hosts indicate a large pool of unprotected systems that can easily be (or already are) part of a large pool of compromised system called *a zombie army* or *botnet*. A market has arisen around renting botnets for only a few cents per system, per week. Spammers, phishers, and other attackers are increasingly using these systems to launch attacks against the global Internet. This trend represents a fundamental shift from attacks against the availability of networks to attacks against individual users.

We believe that these persistently vulnerable hosts will continue to pose a significant threat to the larger Internet community. There is an important question whether it is possible to clean or inoculate this infected population. It is vital that these hosts are quarantined to prevent them from being used as launching points for spamming, phishing, and attacks against critical infrastructure.

6.3.3 Escalated threats

As defender's ability to measure and characterize new Internet threats has grown, attackers have continually been forced to innovate in response. As a result, today's attacks are a far cry from those seen even as recently as five years ago. This trend toward increasingly escalated threats takes several forms:

- Increasingly large and diverse attacker resources. Attackers have huge numbers of compromised computers and resources with which to launch their attacks available to them. Recently, CERT has described botnets with more than 100,000 members, and almost 1 million bot infected hosts have been reported [50, 73]. One fallout of this increased firepower is the ability of attackers to exhaust network resources in attack. Figure 6.9 shows the trend in Denial of Service attack protocols during 2002-2005. Attacks moved away from host-based resource exhaustion attacks, such as SYN floods, to pure network-based resource exhaustion attacks designed to flood bottleneck links. These types of attacks, especially on large, multi-homed networks, are only possible with the massive number of available resources.

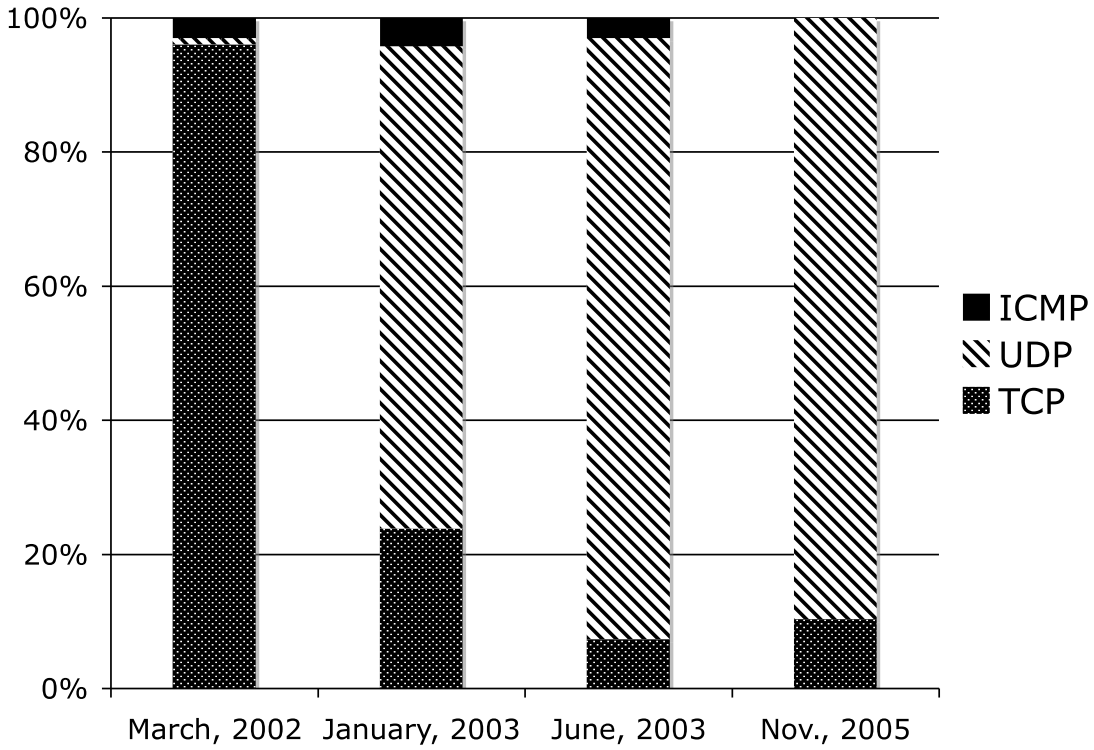


Figure 6.9: The trend in denial of service protocols.

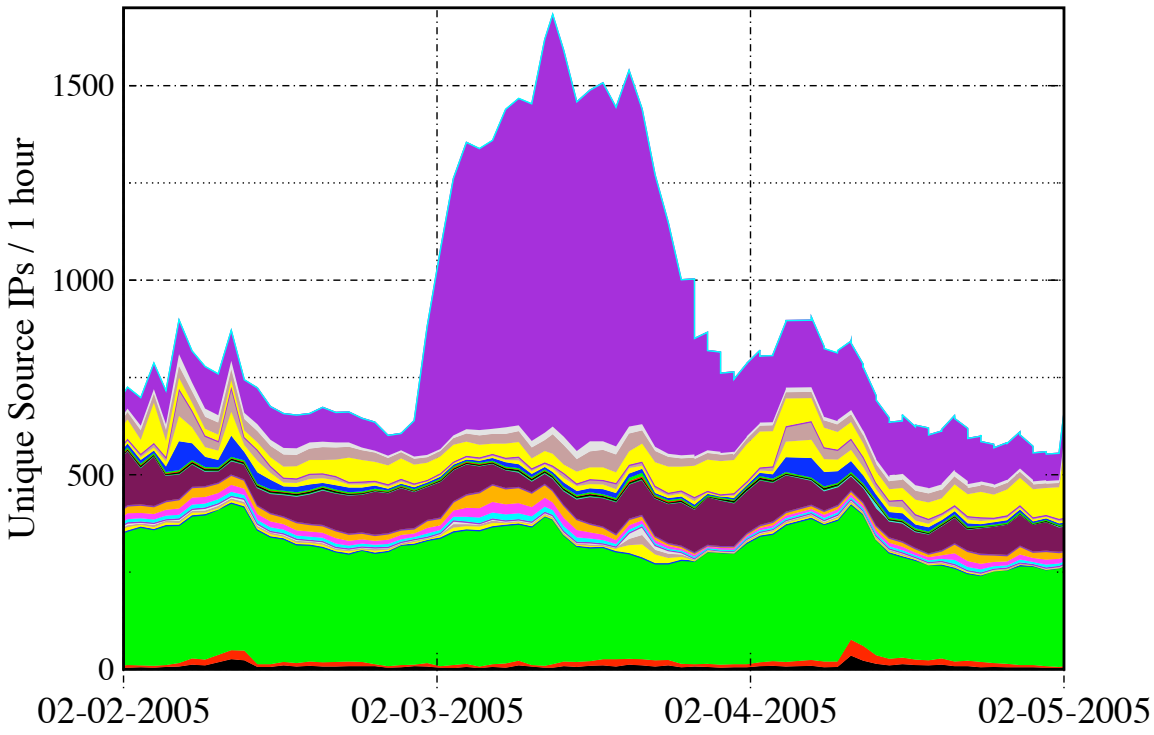


Figure 6.10: A targeted RPC-DCOM attack observed at an academic network containing an IMS sensor.

- Increasingly sophisticated application of resources. In previous years, large-scale DDoS attacks against specific networks were common, and they still persist to this day. What is new is the emergence of large-scale, targeted attacks that seek not to deny service, but to infect a huge number of systems. For example, on January 3rd, 2005 the IMS detected a large attack against a specific academic network involving more than 1,000 attackers attempting to utilize the DCOM RPC vulnerability [74] (Figure 6.10). The intent is not exactly clear, but it was targeted towards a specific Windows service in a specific network. It is well known in the underground community that compromised academic systems are more valuable, due to their high bandwidth and lack of strict filtering policies. Since the attack was targeted at a specific academic network, it is very possible it was an attempt to compromise and control large numbers of academic machines.

- Increasingly complex attacks. Another major shift in Internet threats is that the vulnerability no longer provides much information on the threat. In the past, an infection attempt was used to detect a specific set of threats, such as the CodeRed series of worms or the SQL Slammer worm. Modern threats utilize a large number of propagation mechanisms, making the use of a particular exploit a very poor method of identifying a particular threat. For example, in the Agobot [30] and SDBot [71] worms, the propagation mechanisms are abstracted from the rest of the functionality. The SDBot can propagate using open file shares, p2p networks, backdoors left by previous worms, and by exploiting common Windows vulnerabilities. Due to the modularity and open availability of the source code for many bots, these threats are under constant evolution. SDBot has been reported to have more than 4,000 variants as of August, 2004 [71]. This flexibility makes these threats very hard to classify as they have many of the common features of viruses, worms, and bots.

6.4 Summary

In this chapter, we have shown the utility of the hybrid system by highlighting its ability to characterize, measure, and track Internet threats. We have provided detailed examples of individual threats captured by the system, including the Blaster Internet worm, the SCO Denial of Service attacks, and the Bagle backdoor scanning. We have shown how these and other individual threat analyses can be combined to show a picture of threat trends in the Internet. We highlight several of these trends, including the increasingly distributed population, the persistence of these threats, and

the escalation of threats.

CHAPTER 7

CONCLUSION

The current wave of threats such as worms, denial of service attacks, and botnets pose a significant threat to the availability and security of the Internet. In this thesis we explored the measurement issues associated with tracking and characterizing these threats. We showed how current measurement systems must trade off detailed forensic information, or depth, with large-scale, representative coverage, or breadth. In order to achieve both breadth and depth without any of the corresponding drawbacks, we have proposed a hybrid architecture that combines scalable low-interaction, network-based sensors with high-interaction, host-based sensors. We detailed our efforts in constructioning a distributed network component of the hybrid system and showed the surprising result that different sensors see different perspectives of globally-scoped threats. We argued that the key to the hybrid system's ability to provide both breadth and depth is in its ability to intelligently filter requests seen at the network sensors before handing them to the host-based sensors, and we proposed two novel methods for achieving this. We then examined the role of the host-based sensor in this hybrid architecture and examined two critical problems in host-based operation: how to configure the sensors and how to detect and automate the generation of forensic information. In the next two sections, we summarize some of the key insights our efforts have produced as well as look to what new areas of future work are suggested by our work.

7.1 Insights

First and foremost, this thesis provides insight into the tradeoffs in designing, deploying, and evaluating systems for monitoring Internet threats. We have shown that it is no longer enough to simply monitor one aspect of a threat in order to understand its behavior. While network- and host-

based monitoring system have historically each been successful at providing limited intelligence into the motives and means of attackers, the new threats of today's Internet require systems that can automatically generate actionable information. The emergence of global threats with potentially extreme virulence require that defenders understand the scope of the threats; whether this is a targeted or local attack, as well as quickly detect its emergence. The evolutionary nature of the attacks and the fact that they may occur without time to set specific defenses mean that defenders must have forensic information about the attack that can differentiate good traffic from bad, and differentiate new attacks which may target existing ports or vulnerabilities. In short, network defenders need both breadth and depth in order to combat these threats. The hybrid system presented in this thesis offers the ability to measure, characterize, and track these threats, by making use of intelligent filtering. By providing both breadth and depth, this hybrid system lays the foundation of a variety of important techniques, such as threat mitigation, quarantine, and threat detection.

Our work in network monitoring has provided valuable insights into monitoring non-productive traffic. First, we have shown that darknet traffic, because it is filtered to include only traffic that is non-legitimate, is a scalable means for gaining insight into threats. We have constructed a system that monitors over 1% of routed IP space with a mere handful of resources, a feat not possible if that space contained active hosts. Secondly, we have shown that it is possible to differentiate a great number of these threats in a scalable manner without requiring state. By the use of a transport layer, syn-ack responder, we can illicit the first payloads from a threat without incurring the overhead of connection maintenance. Finally, we have shown that in order to be successful at monitoring Internet threats, a darknet system must not only be sufficiently large, it must also be diverse. We have shown that observations from different sensors vary dramatically in number of monitored packets, source distributions, and destination ports. These sensors continue to show differences, even when monitoring global events with known propagation strategies.

In developing novel intelligent filtering algorithms, we have discovered a variety of properties relevant to the monitoring of threat traffic. We have shown that the darknet traffic seen at individual sensors is dominated by a small fraction of the sources. We have seen that these sources perform the same, small number of tasks repeatedly. We have noted, however, that these sources do not appear with great frequency across sensors in the same time frame. Whether the result of target attacks, or other properties of global threat behavior, again we note that different sensors see different things. Instead of following individual attackers, we make use of our understanding of current threat behaviors, in order to build filtering mechanisms that looks for either the emergence of any new threat

payload, or the rapid increase in the number of attackers performing an attack. These two properties of current threats have shown themselves to be effective in reducing the large number of packets seen at a large distributed monitoring system to a handful of events.

Our work in designing and evaluating host-based sensors has provided us with valuable insight into the the large scale monitoring of events. First, we have dismissed the idea that an effective threat monitoring system can be composed of a monolithic honeyfarm. We have noted that real networks, and real vulnerable populations, are exceptionally diverse, complex, and dynamic, with many different operating system, application, and configuration values. For any honeyfarm system to be successful, it must create representative profiles; hosts that both match to existing systems as well as reflect the relative distributions of the vulnerable population. We have also shown that only observing one type of event on these systems, for example network traffic, is insufficient to fully characterize threat behaviors. However, in providing more detailed information, these host systems, even honeypots, create a surprising amount of information. The use of anomaly detection, however, is an effective means of reducing the number of events to evaluate and in determining when a new event occurs, as these honeypots have no users to introduce entropy.

Finally, the techniques and insight developed here have enabled the construction of a system that has provided us with valuable insights into the threat behaviors themselves. We have been able to characterize a wide range of Internet threats, such as random scanning worms, denial of service attacks, and scanning. We have disseminated these results in operator forums, security lists, and through our own alerting system in an effort to provide real-time actionable information to the operator community. These threat analyses have not only provided a real impact in the security of the network, but taken as a whole have provided us with valuable insight into the evolution of these threats. We have been able to show how the infected populations are becoming increasingly global and diverse. We have shown that the infected populations for worms provide a huge threats to security as large numbers of these hosts remain infected years after the outbreak. Finally, we have been able to characterize the escalation in threats, showing the increase in available resources and the more sophisticated application of those resources. In short, the techniques and insights have enabled us to achieve exactly what we sought – breadth and depth.

7.2 Limitations of the Hybrid Architecture

The measurement framework and architecture presented in this thesis provides a previously unachieved level of breadth and depth. This unique combination allows for robust and timely characterization of current Internet threats. Unfortunately, we do not anticipate that these threat landscapes will remain the same over time as attackers continue to develop new techniques and tools. As such, it is important to understand the potential weaknesses of our architecture as these weakness can and will be exploited by determined attackers. In our current threat model, we concern ourselves with three broad classes of attacks; sensor avoidance, attack obfuscation, and denial-of-service attacks against the architecture.

One potentially harmful avenue of attack is for the attackers to determine the location of the network-based sensors in our sensor network and avoid scanning them. Without these connections appearing in the monitored unused address blocks, the system would not have access to any useful information about these attacks. The key to the success of any such attack is the ability of the attacker to divine the location of these sensors. In our threat model this location information can be discovered in two ways: probe-disclosure, and probe-response. In probe-disclosure the attacker can probe a system using specially crafted messages. When data from the sensor is publicly disclosed, the attacker can decode the message and discover the addresses of the sensors. In probe-response we worry that if a system actively responds to incoming requests to masquerade as a production system, or to elicit collect additional payload data, an attacker can generate a test that differentiates honeypot and real systems. We discuss several approaches to the discover problem in the future work section.

Another avenue of attack against the system involves obfuscation. In this attack class we are concerned that attackers might construct attacks that appear, if not benign, at least less malicious then other attack traffic. This class of attacks effects most prominently the detection and signature generation components of this architecture including attacks against the filtering algorithms, attacks against the host-based sensors. By adding a small amount of random content to each packet the attacker is able to make each packet seem to be unrelated to each other, nullifying the detection of the lightweight content prevalence component. Similarly, with knowledge about the source prevalence algorithm, the attacker can create a slow moving attack whose number of sources never exceeds the historical average. While the presents of two filtering mechanisms provide some resilience to these attacks, they can certainly be combined. In addition the attacker may choose to obfuscate

attack signatures with knowledge about our host-based signature generation. While the simple anomaly detection component offers great resilience to attack as it has a complete picture of all the allowed system activities, this system is limited in its signature generation capabilities. Some small amount of behavioral polymorphism, such as random file generation in random locations, dummy processes, or exceptionally complex behaviors may make it difficult for the system to determine the correct signature for an attack, even if it knows something anomalous is happening.

Finally, the last class of attack we are concerned about in our architecture is attacks against the architecture itself. In its current design, the bottleneck resource in our system is the number and availability of the centralized host based sensors. We envision several attacks that disrupt this resource. First, we envision attacks similar to those discussed above that reduce the effectiveness of filtering. In this instance, however, the goal is not to avoid detection but to exhaust resources. Numerous highly source prevalent attacks against numerous ports, with highly prevalent content could create a run on the host-based resources, creating a denial of service attack. In addition, if the location of the centralized host-based sensors could be discovered, they could be attacked directly, most effectively, but utilizing the inbound bandwidth at some point upstream. Lastly, we envision attacks that make use of vulnerabilities in the various tools we use in our system, allowing attackers to take control of these systems. While a generic maintenance and security issue, it should be noted that the system makes use of several tools, such as VMWare, that attackers develop specialized attack tools to fingerprint and exploit.

7.3 Future Work

One avenue of work unexplored in this thesis is the placement and sizing of sensor blocks. While clearly more monitored address space and more diversity are better, how will we know when we have enough? To a degree, the issue of how much space has been explored by Moore [77]. However, while Moore defines the various parameters of interest, including confidence in detection, vulnerable host populations, seeded hosts, scanning rate, etc., he comes to no firm conclusion about the size needed. An examination of the threat landscape, the vulnerable population distribution, and current mitigation techniques is required. To complicate matters further, the size is tied closely with the number of and placement of the diverse blocks. Items such as policy, resource constraints, and proximity to infected population play a role in the visibility of these sensors [32]. Determining an algorithm for placement must take into account these factors as well as the size of the monitored

blocks in determining how many blocks are required and in what locations.

In this thesis we have examined the use of the hybrid system as a novel point in the spectrum, which trades off breadth for depth. While we have demonstrated the utility of this system across a wide range of threats, our unique point in this spectrum allows for the analysis of two different threats that have not been covered in this thesis; Botnets and misconfiguration. As the hybrid system as enabled detailed forensic information about threats, its has begun to uncover a wealth of interesting information about the various components of the bot life cycle, including its propagation, communication, and attack behaviors [34]. In addition, as the system is better able to classify threat behaviors into types, such as denial of service attacks, worms, and botnets, we are being left with a sizeable amount of unclassified events that appear to be misconfigurations.

Another fundamental challenge for all honeypot/darknet Internet sensor systems is the possibility that an attacker could fingerprint and then avoid the sensor, or pollute the data collected. In our current threat model, we worry about two basic methods whereby an attacker can discover detection systems. In Probe-Disclosure the attacker can probe a darknet system using specially crafted messages. When data from the sensor is publicly disclosed, the attacker can decode the message and discover the addresses of the sensors [20]. For example, if a passive darknet system publicly reported all the packet payloads it received each day, an attacker could send a packet to every IPv4 address on the Internet with a payload containing the address of the sensor. The attacker can then simply inspect all the published packet payloads and recover the addresses of the darknet sensors. One approach to this problem would be to eliminating sensors specific data from published traces. By publishing only that data which is prevalent accross multiple sensors, we elinate many of the hidden channels an attacker may use. Of course, another soultion is to limit the published information to trusted sources. In Probe-Response we worry that if a darknet system actively responds to incoming requests to masquerade as a production system, or to elicit collect additional payload data, an attacker can generate a test which differentiates honeypot and real systems [72]. For example, if the darknet responds to request on TCP port 80, an attack would look for valid HTTP responses. One approach to solving this problem is to build consistent and representative configurations for the network systems. By constucting hosts and application distributions which match those of the surrounding production systems, we make it more difficult for the attackers to deduce the presence of anomalous systems.

One limitation of the current system is its reliance on the scanning behaviors of threats in order to observe their behavior. As these threats become more targeted (see Chapter 6), the worms

begin to use topological propagation [108], or require user interaction for propagation instead of scanning [123] the utility of such as system begins to diminish. In order to combat these newer threats we need to investigate ways to make these systems emulate the necessary human behaviors and construct more representative honeypots. Techniques such as seeding email addresses for email honeypots [99], building honeypots that shadow real user connections [9], and using bots to download potential malicious content to honeypots [122] all offer potential avenues for bringing these new threats to the honeypots for analysis.

Finally, while we have clearly focused our efforts on examining attacks that are global in nature, affecting thousands of autonomous systems. One promising are of future work is to instead, characterize the threats to individual enterprises. If darknets could be ubiquitously deployed within an enterprise [31], how would that visibility change not only our perspective of global threats, but of local, targeted attacks as well? Initial deployments of this ubiquitous system have shown that the darknet traffic seen inside an enterprise can vary greatly in quantity and substance from individual public blocks in the same organization. The reasons for these differences and the impact of the ubiquitous monitoring remain an open research issue.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] *Bait-N-Switch Honeypot*. http://violating.us/projects/baitnswitch/.

[2] *KFSensor*. http://www.keyfocus.net/kfsensor/.

[3] *NetBait*. http://www.netbaitinc.com/products/products.html.

[4] *NetFacade*. http://www22.verizon.com/fns/netsec/fns_netsecurity_netfacade.html.

[5] *Specter*. http://www.specter.com.

[6] *CERT Incident Note IN-2004-01: W32/Novarg.A Virus*. `http://www.cert.org/incident_notes/IN-2004-01.html`, 2004.

[7] *Symantec Security Response - W32.Beagle.A*. `http://securityresponse.symantec.com/avcenter/venc/data/w32.beagle.a@mm.html`, 2004.

[8] G. Almes. *Metrics and Infrastructure for IP Performance*. `http://www.advanced.org/surveyor/`, September 1997.

[9] 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*, Baltimore, MD, August 2005.

[10] Anti-Phishing Working Group. *Phish Activity Trends Report*. `http://www.antiphishing.org/`, January 2005.

[11] I. Arce and E. Levy, "An analysis of the Slapper Worm," *IEEE Security & Privacy*, vol. 1, no. 1, pp. 82–87, January/February 2003.

[12] M. Aron, P. Druschel, and W. Zwaenepoel. *Efficient Support for P-HTTP in Cluster-Based Web Servers*, June 04 1999.

[13] M. Bailey, E. Cooke, T. Battles, and D. McPherson. *Tracking Global Threats with the Internet Motion Sensor*. 32nd Meeting of the North American Network Operators Group, October 2004.

[14] M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson, "The Internet Motion Sensor: A distributed blackhole monitoring system," in *Proceedings of Network and Distributed System Security Symposium (NDSS '05)*, San Diego, CA, February 2005.

[15] M. Bailey, E. Cooke, F. Jahanian, and N. Provos, "A hybrid honeypot architecture for scalable network monitoring," Technical Report CSE-TR-499-04, University of Michigan, 2004.

[16] M. Bailey, E. Cooke, F. Jahanian, N. Provos, K. Rosaen, and D. Watson, "Data Reduction for the Scalable Automated Analysis of Distributed Darknet Traffic," *Proceedings of the USENIX/ACM Internet Measurement Conference*, October 2005.

[17] M. Bailey, E. Cooke, D. Watson, F. Jahanian, and J. Nazario, "The Blaster Worm: Then and Now," *IEEE Security & Privacy*, vol. 3, no. 4, pp. 26–31, 2005.

[18] L. Baldwin. *My Net Watchman - Network Intrusion Detection and Reporting*. `http://www.mynetwatchman.com/`, 2005.

[19] T. Berners-Lee, R. Fielding, and H. Frystyk. *RFC 1945 - Hypertext Transfer Protocol – HTTP/1.0*. http://www.faqs.org/rfcs/rfc1945.html, May 1996.

[20] J. Bethencourt, J. Franklin, and M. Vernon, "Mapping Internet sensors with probe response attacks," in *Proceedings of the 14th USENIX Security Symposium*, Baltimore, MD, August 2005.

[21] F. Castaneda, E. C. Sezer, and J. Xuy, "Worm vs. worm: Preliminary study of an active counter-attack mechanism," in *Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM-04)*, New York, October 29 2004, ACM Press.

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

[23] CERT. *Code Red II: Another Worm Exploiting Buffer Overflow In IIS Indexing Service DLL*. `http://www.cert.org/incident_notes/IN-2001-09.html`, August 2001.

[24] CERT. *CERT Advisory CA-2003-20 W32/Blaster worm*. `http://www.cert.org/advisories/CA-2003-20.html`, August 2003.

[25] CERT Coordination Center, "CERT Advisory CA-2003-20 W32/Blaster worm,", August 2003.

[26] CERT Coordination Center, Carnegie Mellon University. *CERT/CC Overview Incident and Vulnerability Trends*. `http://www.cert.org/present/cert-overview-trends/`, 2003.

[27] B. Cheswick, "An evening with Berferd in which a cracker is lured, endured, and studied," in *Proceedings of the Winter 1992 USENIX Conference: January 20 — January 24, 1992, San Francisco, California*, pp. 163–174, Berkeley, CA, USA, Winter 1992.

[28] F. Cohen, *A Short Course on Computer Viruses*, John Wiley & Sons, 2nd edition, April 1994.

[29] F. Cohen. *The Deception Toolkit (DTK)*. `http://www.all.net/dtk/dtk.html`, June 2004.

[30] Computer Associates. *Win32.Agobot*. `http://www3.ca.com/securityadvisor/virusinfo/virus.aspx?id=37776`, July 2004.

[31] E. Cooke, M. Bailey, F. Jahanian, and R. Mortier, "The dark oracle: Perspective-aware unused and unreachable address discovery," in *Proceedings of the 3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI '06)*, May 2006.

[32] E. Cooke, M. Bailey, Z. M. Mao, D. Watson, and F. Jahanian, "Toward understanding distributed blackhole placement," in *Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM-04)*, New York, Oct 2004, ACM Press.

[33] E. Cooke, M. Bailey, D. Watson, F. Jahanian, and J. Nazario, "The Internet motion sensor: A distributed global scoped Internet threat monitoring system," Technical Report CSE-TR-491-04, University of Michigan, Electrical Engineering and Computer Science, July 2004.

[34] E. Cooke, F. Jahanian, and D. McPherson, "The Zombie roundup: Understanding, detecting, and disrupting botnets," in *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet (SRUTI 2005 Workshop)*, Cambridge, MA, July 2005.

[35] E. Cooke, Z. M. Mao, and F. Jahanian, "Worm hotspots: Explaining non-uniformity in worm targeting behavior," Technical Report CSE-TR-503-04, University of Michigan, 2004.

[36] M. Corporation. *Unchecked Buffer in Index Server ISAPI Extension Could Enable Web Server Compromise*. http://www.microsoft.com/technet/security/bulletin/MS01-033.mspx, June 2001.

[37] M. Corporation. *What You Should Know About the Blaster Worm*. `http://www.microsoft.com/security/incident/blast.mspx`, August 2003.

[38] M. Corporation. *Microsoft Build Alliance With Service Providers to Advance Security of Internet Users Worldwide*. `http://www.microsoft.com/presspass/press/2004/feb04/02-24GIAISpr.asp`, February 24 2004.

[39] M. Corporation. *What You Should Know About the Sasser Worm*. `http://www.microsoft.com/security/incident/sasser.mspx`, May 2004.

[40] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, "Vigilante: End-to-End containment of Internet worms," in *Proceedings of the ACM Symposium on Operating System Principles*, October 2005.

[41] T. Cymru. *The Darknet Project*. `http://www.cymru.com/Darknet/index.html`, June 2004.

[42] D. Dagon, X. Qin, G. Gu, J. Grizzard, J. Levine, W. Lee, and H. Owen, "Honeystat: Local worm detection using honeypots," in *Recent Advances in Intrusion Detection, 7th International Symposium, (RAID 2004)*, Lecture Notes in Computer Science, Sophia-Antipolis, French Riviera, France, October 2004, Springer.

[43] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1*. http://www.faqs.org/rfcs/rfc2616.html, June 1999.

[44] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, "A sense of self for Unix processes," in *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pp. 120–128, Oakland, CA, May 1996.

[45] A. L. Foster, "Colleges Brace for the Next Worm," *The Chronicle of Higher Education*, vol. 50, no. 28, , 2004.

[46] Fyodor. *Nmap*. http://www.insecure.org.

[47] B. R. Greene and D. McPherson. *Sinkholes: A Swiss Army Knife ISP Security Tool*. `http://www.arbor.net/`, June 2003.

[48] HD Moore. *DCOM RPC exploit (dcom.c)*. http://lists.netsys.com/pipermail/full-disclosure/2003-July/007092.html, July 2003.

[49] J. L. Hellerstein, F. Zhang, and S. Shahabuddin, "A statistical approach to predictive detection," *Computer Networks (Amsterdam, Netherlands: 1999)*, vol. 35, no. 1, pp. 77–95, January 2001.

[50] A. Householder and R. Danyliw, "CERT Advisory CA-2003-08 Increased Activity Targeting Windows Shares,", 2003.

[51] C. S. Inc. *NetFlow Services and Applications*. http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.htm, 2002.

[52] Internet Systems Consortium, Inc. *ISC Internet Domain Survey*. http://www.isc.org/, 2004.

[53] *Internet Performance Measurement and Analysis (IPMA) project homepage*. http://www.merit.edu/ipma/.

[54] *IP Performance Metrics (IPPM)*. http://www.ietf.org/html.charters/ippm-charter.html.

[55] C. Ji and M. Thottan, "Adaptive thresholding for proactive network problem detection," in *Third IEEE International Workshop on Systems Management*, pp. 108–116, Newport, Rhode Island, April 1998.

[56] X. Jiang and D. Xu, "Collapsar: A VM-based architecture for network attack detention center," in *Proceedings of the 13th USENIX Security Symposium*, San Diego, CA, USA, August 2004.

[57] H.-A. Kim and B. Karp, "Autograph: Toward Automated, Distributed Worm Signature Detection," in *Proceedings of the 2004 USENIX Security Symposium*, San Diego, CA, USA, August 2004.

[58] S. T. King and P. M. Chen, "Backtracking intrusions," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03)*, pp. 223–236, Bolton Landing, NY, USA, October 2003, ACM.

[59] C. Kreibich and J. Crowcroft, "Honeycomb: creating intrusion detection signatures using honeypots," in *2nd Workshop on Hot Topics in Networks (HotNets-II)*, Boston, MA, 2003.

[60] J. Kristoff. *Botnets*. 32nd Meeting of the North American Network Operators Group, October 2004.

[61] A. Kumar, V. Paxson, and N. Weaver, "Exploiting underlying structure for detailed reconstruction of an internet-scale event," *Proceedings of the USENIX/ACM Internet Measurement Conference*, October 2005.

[62] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515–528, October 1998.

[63] Lad, Zhao, Zhang, Massey, and Zhang, "Analysis of BGP update surge during slammer worm attack," in *International Workshop on Distributed Computing: Mobile and Wireless Computing, LNCS*, volume 5, 2003.

[64] C. Leckie and R. Kotagiri, "A probabilistic approach to detecting network scans," in *Proceedings of the Eighth IEEE Network Operations and Management Symposium (NOMS 2002)*, pp. 359–372, Florence, Italy, April 2002.

[65] R. Lemos. *MySQL worm halted.* http://ecoustics-cnet.com.com/MySQL+worm+halted/2100-7349_3-5555242.html, January 2005.

[66] T. Liston. *LaBrea - Homepage*. http://labrea.sourceforge.net/.

[67] J. Mahdavi, M. Mathis, and V. Paxson, "Creating a national measurement infrastructure (NIMI)," in *Proceedings of Internet Statistics and Metrics Analysis (ISMA) '97*, May 1997.

[68] G. R. Malan and F. Jahanian, "An extensible probe architecture for network protocol performance measurement," in *Proceedings of ACM SIGCOMM*, pp. 177–185, Vancouver, British Columbia, September 1998.

[69] Z. Mao, R. Bush, T. Griffin, and M. Roughan, "BGP Beacons," in *Proc. Internet Measurement Workshop*, October 2003.

[70] J. S. Marcus, "AS Number Exhaustion," in *Proceedings of the 28th RIPE Meeting*, Amsterdam, the Netherlands, January 2001.

[71] McAfee. *W32/Sdbot.worm.* http://vil.nai.com/vil/content/v_100454.htm, April 2003.

[72] B. McCarty, "The honeynet arms race," *Security and Privacy Magazine, IEEE*, vol. 1, no. 6, pp. 79 – 82, Nov. 2003.

[73] L. McLaughlin, "Bot software spreads, causes new worries," *IEEE Distributed Systems Online*, vol. 5, no. 6, , June 2004.

[74] Microsoft. *DCOM RPC vulnerability*. http://www.microsoft.com/technet/security/bulletin/MS03-026.mspx, July 2003.

[75] D. Moore, "Network telescopes: Observing small or distant security events," in *11th USENIX Security Symposium, Invited talk*, San Francisco, CA, August 5–9 2002. Unpublished.

[76] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the Slammer worm," *IEEE Security & Privacy*, vol. 1, no. 4, pp. 33–39, 2003.

[77] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Network telescopes," Technical Report CS2004-0795, UC San Diego, July 2004.

[78] D. Moore, G. M. Voelker, and S. Savage, "Inferring Internet denial-of-service activity," in *Proceedings of the Tenth USENIX Security Symposium*, pp. 9–22, Washington, D.C., August 2001.

[79] J. Nazario, *Defense and detection strategies against Internet worms*, Artech, 2004.

[80] J. Nazario, M. Bailey, and F. Jahanian, "The spread of the Blaster worm,".

[81] T. L. S. of Delirium Research Group. *Buffer Overrun in Windows-RPC Interface*. http://lsd-pl.net/special.html, 2003.

[82] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson, "Characteristics of Internet background radiation," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pp. 27–40. ACM Press, 2004.

[83] K. Peterson, "Blaster hacker receives 18-month sentence," *Seattle Times*, vol. Saturday, January 29, , 2005.

[84] N. Provos. *The Honeyd Project*, June 2004.

[85] N. Provos, "Honeyd — A virtual honeypot daemon," in *10th DFN-CERT Workshop*, Hamburg, Germany, February 2003.

[86] N. Provos, "A Virtual Honeypot Framework," in *Proceedings of the 13th USENIX Security Symposium*, pp. 1–14, San Diego, CA, USA, August 2004.

[87] M. Ranum et al. *NFR security resource center: back officer friendly*. `http://www.nfr.com/resource/backOfficer.php`.

[88] E. Rescorla, "Security holes... Who cares?," in *Proceedings of USENIX Security Symposium*, August 2003.

[89] S. S. Response. *W32.Randex.E*. `http://securityresponse.symantec.com/avcenter/venc/data/w32.randex.e.html`, 2003.

[90] S. S. Response. *W32.Welchia.Worm*. `http://securityresponse.symantec.com/avcenter/venc/data/w32.welchia.worm.html`, 2003.

[91] Reuters. *Bank of America ATMs disrupted by virus*. `http://archive.infoworld.com/togo/1.html`, January 2003.

[92] Robert Graham. *Decompiled Source for MS RPC DCOM Blaster Worm*. `http://robertgraham.com/journal/030815-blaster.c`, 2003.

[93] P. Roberts, "Suspected blaster-f author nabbed," *PC World*, vol. Wednesday, September 03, , 2003.

[94] S. Robertson, E. Siegel, M. Miller, and S. Stolfo, "Surveillance detection in high bandwidth environments," in *Proceedings of the Third DARPA Information Survivability Conference and Exposition (DISCEX 2003)*. IEEE Computer Society Press, April 2003.

[95] M. Roesch, "Snort — lightweight intrusion detection for networks," in *Proceedings of the Thirteenth Systems Administration Conference (LISA XIII): November 7—12, 1999, Seattle, WA, USA*, USENIX, editor, Berkeley, CA, USA, 1999, USENIX.

[96] A. Ron, D. Plonka, J. Kline, and P. Barford, "A signal analysis of network traffic anomalies," *Internet Measurement Workshop 2002*, August 09 2002.

[97] SANS. *SANS - Internet Storm Center - Cooperative Cyber Threat Monitor and Alert System*. `http://isc.incidents.org/`, June 2004.

[98] S. Savage, G. Voelker, G. Varghese, V. Paxson, and N. Weaver. *Center for Internet Epidemiology and Defenses*. NSF CyberTrust Center Proposal, 2004.

[99] G. Schryen, "An e-mail honeypot addressing spammers' behavior in collecting and applying addresses," in *Proceedings of the IEEE International Workshop on Information Assurance*, West Point, NY, 2005.

[100] C. Shannon and D. Moore, "The spread of the Witty worm," *IEEE Security & Privacy*, vol. 2, no. 4, pp. 46–50, July/August 2004.

[101] C. Shannon, D. Moore, and J. Brown, "Code-Red: a case study on the spread and victims of an Internet worm," in *Proceedings of the Internet Measurement Workshop (IMW)*, December 2002.

[102] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *6th Symposium on Operating Systems Design and Implementation (OSDI '04)*, pp. 45–60, San Francisco, CA, December 6–8 2004.

[103] D. Song, R. Malan, and R. Stone. *A Snapshot of Global Internet Worm Activity*. FIRST Conference on Computer Security Incident Handling and Response, June 2002.

[104] D. Song, R. Malan, and R. Stone. *A Snapshot of Global Internet Worm Activity*. Arbor Network Technical Report, June 2002.

[105] E. H. Spafford, "The internet worm program: An analysis," Technical Report Purdue Technical Report CSD-TR-823, West Lafayette, IN 47907-2004, 1988.

[106] L. Spitzner, *Honeypots: Tracking Hackers*, Addison-Wesley, 2002.

[107] L. Spitzner et al. *The Honeynet Project*. `http://project.honeynet.org/`, June 2004.

[108] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*. USENIX, August 2002.

[109] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison Wesley, 1994.

[110] J. Stewart. *[Full-Disclosure] [Fwd: [TH-research] Bagle remote uninstall]*. `http://lists.netsys.com/pipermail/full-disclosure/2004-January/015970.html`, January 2004.

[111] R. Stone, "CenterTrack: An IP overlay network for tracking DoS floods," in *9th Usenix Security Symposium*, August 2000.

[112] Symantec Corporation. *Symantec Decoy Server (a.k.a ManTrap)*. `http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=157&EID=0`, June 2004.

[113] Symantec Corporation. *DeepSight Analyzer*. `http://analyzer.securityfocus.com/`, 2005.

[114] U.-C. U. S. C. E. R. Team. *US-CERT Cyber Security Bulletin*. `http://www.us-cert.gov/cas/body/bulletins/SB04-147.pdf`, 2004.

[115] U.-C. U. S. C. E. R. Team. *W32Mydoom.C or W32.HLLW.Doomjuice*. `http://www.us-cert.gov/current/archive/2004/02/13/archive.html`, 2004.

[116] I. Trend Micro. *WORM AGOBOT.GEN - Description and solution*. http://www.trendmicro.com/vinfo/virusencyclo/default5.asp?VName=WORM_AGOBOT.GEN, 2003.

[117] J. Ullrich. *DShield*. http://www.dshield.org, 2000.

[118] United Nations Conference on Trade and Development. *E-Commerce and Development Report*. http://www.unctad.org/Templates/webflyer.asp?intDocItemID=10029&docid=5633&intItemID=3356&lang=1&mode=press, December 2004.

[119] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage, "Scalability, fidelity and containment in the Potemkin virtual honeyfarm," in *Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, Brighton, UK, October 2005.

[120] B. Walters, "VMware virtual platform," *j-LINUX-J*, vol. 63, pp. ??–??, July 1999.

[121] H. Wang, C. Guo, D. Simon, and A. Zugenmaier, "Shield: Vulnerability-driven network filters for preventing known vulnerability exploits," in *Proc of ACM SIGCOMM*, Portland, Oregon, USA, 2004.

[122] Y.-M. Wang, D. Beck, X. Jiang, and R. Roussev, "Automated web patrol with strider honeymonkeys: Finding web sites that exploit browser vulnerabilities," Technical Report MSR-TR-2005-72, Microsoft Research (MSR), August 2005.

[123] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham, " A Taxonomy of Computer Worms," in *Proc of ACM CCS Workshop on Rapid Malcode*, pp. 11–18. ACM Press, October 2003.

[124] 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 Symposium on Network and Distributed Systems Security (NDSS 2004)*, San Diego, CA, February 2004, Internet Society.

[125] V. Yegneswaran, P. Barford, and S. Jha, "Global intrusion detection in the DOMINO overlay system," in *Proceedings of Network and Distributed System Security Symposium (NDSS '04)*, San Diego, CA, February 2004.

[126] V. Yegneswaran, P. Barford, and D. Plonka, "On the design and use of Internet sinks for network abuse monitoring," in *Recent Advances in Intrusion Detection—Proceedings of the 7th International Symposium (RAID 2004)*, Sophia Antipolis, French Riviera, France, October 2004.

[127] C. 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*, pp. 190–199. ACM Press, 2003.