Context-Aware Network Security

by

Sushant Sinha

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctoral of Philosophy
(Computer Science and Engineering)
in The University of Michigan
2009

Doctoral Committee:

Professor Farnam Jahanian, Chair
Professor Kang G. Shin
Assistant Professor Rahul Sami
Research Scientist Michael D. Bailey
Associate Professor Jignesh M. Patel, University of Wisconsin

To my father Arun Kumar Sinha and mother Mira Sinha.

ACKNOWLEDGEMENTS

First and foremost I would like to thank my advisor Farnam Jahanian for the exciting and fruitful research that we have done at the University of Michigan. He invested a significant amount of time and effort discussing research problems and critically questioning my research ideas. His unrelenting support for exploring new ideas was of great importance to me. I will always remember Farnam's emphasis on going the extra mile to show the practical utility of a research project.

I thank Michael Bailey for working with me on a number of research projects. His ability to quickly distill important ideas from research projects is truly outstanding, and I am still learning from him. I would like to thank Jignesh Patel for interesting discussions on performance improvement of software systems. I would also like to thank my committee members Kang G. Shin and Rahul Sami for going through the dissertation and providing me useful feedback.

Additionally, I would like to thank Evan Cooke for his help with research papers and in demonstrating the importance of passion in one's work. I would also like to thank my office mates Jon Oberheide, Mona Attarian, Dan Peek, Benjamin Wester, Ya-Yunn Su, Manish Anand, Ed Nightangale, Kaushik Veeraghavan, Eric Vander Weele, Kaustubh Nyalkalkar, Yunjing, and Thomas Holland in providing an excellent environment for discussion and debates.

Last, but not the least, I thank my family for supporting my "never ending" Ph.D journey, and my wife Khushbu for coping with my final, yet busy, days of dissertation.

TABLE OF CONTENTS

LIST OF TABLES

Table

LIST OF FIGURES

Figure

ABSTRACT

Context-Aware Network Security

by

Sushant Sinha

Chair: Farnam Jahanian

The rapid growth in malicious Internet activity, due to the rise of threats like automated worms, viruses, and botnets, has driven the development of tools designed to protect host and network resources. One approach that has gained significant popularity is the use of network based security systems. These systems are deployed on the network to detect, characterize and mitigate both new and existing threats.

Unfortunately, these systems are developed and deployed in production networks as *generic* systems and little thought has been paid to customization. Even when it is possible to customize these devices, the approaches for customization are largely manual or ad hoc. Our observation of the production networks suggest that these networks have significant diversity in end-host characteristics, threat landscape, and traffic behavior – a collection of features that we call the *security context* of a network. The scale and diversity in security context of production networks make manual or ad hoc customization of security systems difficult. Our thesis is that automated adaptation to the security context can be used to significantly improve the performance and accuracy of network-based security systems.

In order to evaluate our thesis, we explore a system from three broad categories of network-based security systems: known threat detection, new threat detection, and reputation-based mitigation. For known threat detection, we examine a signature-based intrusion detection system and show that the system performance improves significantly if it is aware of the signature set and the traffic characteristics of the network. Second, we explore a large collection of honeypots (or honeynet) that

are used to detect new threats. We show that operating system and application configurations in the network impact honeynet accuracy and adapting to the surrounding network provides a significantly better view of the network threats. Last, we apply our context-aware approach to a reputation-based system for spam blacklist generation and show how traffic characteristics on the network can be used to significantly improve its accuracy.

We conclude with the lessons learned from our experiences adapting to network security context and the future directions for adapting network-based security systems to the security context.

CHAPTER 1

Introduction

The Internet has significantly simplified public communication and enabled a variety of electronic activities such as shopping, banking, reading blogs, and social networking. Many of these activities require end-users to divulge personal information, which is then automatically stored and processed by software components. With the increasingly ubiquitous nature of the Internet and the large number of vulnerabilities in software systems, the Internet has become an easy medium for information theft and abuse. Threats in the form of worms, viruses, botnets, spyware, spam, and denial of service [13, 19, 78, 48] are rampant on today's Internet.

To counter these threats, a number of systems have been developed to protect host and network resources. The host-based security systems have visibility into the filesystem, processes, and network access on the machine at a very fine level of granularity. However, they require instrumenting end-hosts, and that can adversely impact the availability and performance of a production service. On the other hand, network-based security systems are easy to deploy because they require little or no modifications to the end-hosts. They receive an aggregated view of the network but have limited visibility into the end-hosts.

Due to the ease of deployment, network-based security systems have gained significant popularity in recent years. They can be broadly classified into three categories, i.e., *known threat detection systems*, *new threat detection systems*, and *reputation-based detection systems*. Known threat detection systems generally use signatures of known threats and include systems like Nessus [5] to detect vulnerable network services and Snort [67] to detect and mitigate known intrusions. New threat detection systems include anomaly detection systems and unused resource monitoring systems such as

honeypots (e.g., honeyd [63]). Reputation-based systems use some measure of good or bad activity and then use it to detect new threats. These include domain blacklisting like Google Safe Browsing API [38] and spam blacklists like SpamHaus [11]. Since reputation-based systems use some known way to detect good or bad and then use the reputation to detect future threats, they fall somewhere in between the known threat detection systems and the new threat detection systems.

While network-based security systems themselves have improved considerably over time, they still take a *"one-size fits all"* approach when deployed in different networks. These systems are typically viewed as generic solutions and fail to leverage the contextual information available in the networks to customize their deployment. Unfortunately, this information may be critical to the performance and accuracy of these systems as the networks they are deployed in differ significantly with each other in terms of policy, the topological layout, the vulnerability landscape, the exploits observed, the traffic characteristics, etc.

Many network-based security systems acknowledge the need to adapt to the network. However, such adaptation is often decided in an ad hoc fashion or left to be manually configured. For example, honeypot systems like honeyd [63] come with a default configuration file for the operating system and the vulnerability configuration of the honeypots. While such systems can be manually configured by a network administrator, the scale of configuration and the diversity among different networks make it very challenging. For example, configuring honeynet in a network may require one to come up with operating system and application configuration for thousands of hosts. In addition, the diversity among networks make it difficult for people to share their configurations and mitigate this effort.

Our thesis is that automatic adaptation to the network context will significantly improve the performance and accuracy of network-based security systems. In order to evaluate our thesis, we explore a system from three broad categories of network-based security systems: known threat detection, new threat detection, and reputation-based mitigation. For known threat detection, we examine a signature-based intrusion detection system and show that the system performance improves significantly if it is aware of the signature set and the traffic characteristics of the network. Second, we explore a large collection of honeypots (or honeynet) that are used to detect new threats. We show that operating system and application configurations in the network impact honeynet accuracy and adapting to the surrounding network provides a significantly better view of the network

threats. Last, we apply our context-aware approach to a reputation-based system for spam blacklist generation. We show how traffic characteristics on the network impact its accuracy and then develop a technique for automatically adapting the system to the traffic characteristics.

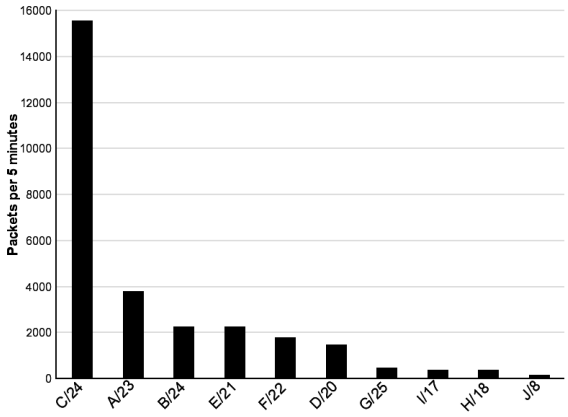
While we do not propose an *automated* way to adapt any network-based security system, we lay down a rigorous path for analyzing the impact of context on the performance and accuracy of the system and then developing techniques that automatically take the context into account. The extensive studies that we present lay down the foundation for adapting a new security system to a given network context.

Adding context to computing so that they can better serve human needs has received significant attention from the academic community. Explicitly adding context for each user is too cumbersome and so most of these attempts have tried to leverage user group activities to automatically infer the context. In the mobile computing community, context in the form of location, role and time has been used to automatically adapt a mobile device [52]. For example, a cell phone can vibrate instead of ringing if it knows that a person is in a meeting. Search engine technologies have leveraged user clicks to determine the context for a query and improve search ranking [42]. They do not take explicit feedback from the users but automatically determine the context for a query by exploiting the large amount of click-through data.

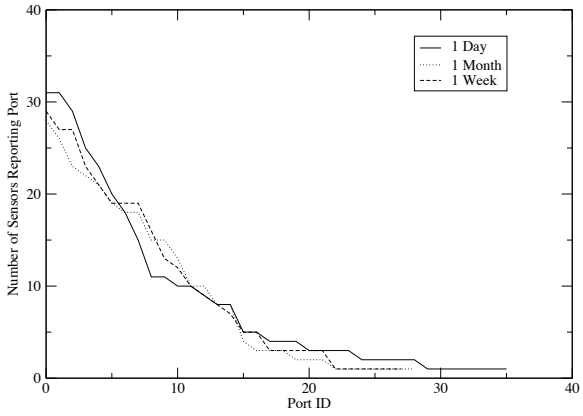
In the following sections, we first explain what is meant by the security context of a network and argue that security context changes significantly over time and space. We then provide a brief overview of our techniques to automatically adapt three types of network-based security systems and show how these techniques significantly improve the performance and accuracy of these systems.

1.1 Context

Context can mean a variety of things to computer scientists, from one's physical location, to one's current desktop environment, to the task being performed, etc. We begin, therefore, by providing a definition of what we mean by context and more specifically, context-aware security. We show that the properties important to context-aware security are, in fact, non-trivial to measure. However, we argue that if they can be measured, context may be used in a variety of ways that improve performance and accuracy of existing and new security applications.



(a)



(b)

Figure 1.1: (a) Packet rate as seen by each sensor normalized by /24 (from [24].) (b) The number of darknets (of 31) reporting a port in the top 10 ports over a day, week, and month time frame (from [17].). Different networks have different attack surfaces.

1.1.1 What is Security Context?

Short of unplugging our computers from the network and locking them in a room, there is no absolute security. At its most fundamental level, the security is a risk analysis activity in which practitioners decide what they wish to protect from whom and at what cost. The key to understanding these tradeoffs are three properties, which we define to make up a network's security context:

- **Vulnerability profile:** The vulnerability profile represents the space of all possible targets and ideally all methods of unauthorized access to those services. In the traditional sense, this is a mapping between the device (i.e., machine), operating system, applications, and the list of known vulnerabilities for each. More broadly, this encompasses unknown vulnerabilities in server software and the social engineering path for access acquisition in client software.

- **Attack surface:** The attack surface represents the unique threats posed by attackers to the defenders of a particular network. In a traditional sense, it is a measure of the remote network exploits (either attempted or successful) directed at a particular network. In a broader sense, it encompasses the notion of who the attackers are, what resources they are interested in, and the current techniques for acquiring those resources. For example, while a network might run a large number of (potentially vulnerable) printer services, attackers may avoid these services due to their uniqueness (and hence difficulty in exploiting), as well as the limited value in compromising them. Of course, other attackers may feel just the opposite about having access to printed documents – the context matters.

- **Usage model:** While the attack surface helps prioritize the potential targets specified in the vulnerability surface by defining what the attackers are interested in and the current tools used to achieve them, the usage model helps defenders prioritize the importance of the services on the network. This prioritization may be as simple as defining what services are most used on a network, but may layer in notions of data importance, disclosure liability, opportunity costs on failure in availability, etc.

1.1.2 Does Security Context Change?

In the previous section, we defined security context to include a network's attack surface, its vulnerability profile, and its usage model. However, before we talk about adaptation of security

Operating System	Networks			
	A/16	B/16	C#1/17	D#1/19
Windows	44	25	76	77
Cisco IOS	14	7	-	-
Apple	9	36	-	-
Linux	9	7	15	6
.HP printer	3	13	-	-
Solaris	9	7	1	2
*BSD	1	-	8	15

Operating Systems

TCP Port	Networks			
	A/16	B/16	C#1/17	D#1/19
139	42	17	-	-
22	41	53	30	25
135	39	10	42	69
23	27	34	4	5
445	27	11	-	-
80	21	26	93	96
25	12	10	70	83
21	8	24	77	79
427	4	26	-	-
497	3	28	-	-
110	1	-	39	17

TCP ports

Table 1.1: Comparing the vulnerable population in four networks, by operating systems and TCP ports(from [72]). Different networks have different vulnerability profiles.

Hospital			Library Regional Network			Government			Small College		
APPL.	IN	OUT	APPL.	IN	OUT	APPL.	IN	OUT	APPL.	IN	OUT
RTSP	96.25	1.91	HTTP	17,590	2,200	HTTP	1,390	231.43	HTTP	58,420	13,710
DNS	1.69	1.85	HTTPS	651.6	116.39	SSH	11.85	195.87	FLASH	4,080	84.75
HTTPS	3.49	.001	FLASH	706.02	13.13	HTTPS	108.49	98.11	HTTPS	1,280	1,400
SMTP	1.99	1.32	TCP/81	16.41	287.16	ESP	98.97	32.64	XBOX	1,010	1,610
LOTUS NOTES	1.73	.158	SMTP	100.83	166.51	SMTP	55.72	73.00	UNIDATA -LDM	947.91	950.88

Table 1.2: The network application usage (Kbps) at 4 different networks. Different networks have different usage models.

device to the security context it is important to determine whether the security context changes. In this section, we examine the two main reasons for context change: the diversity among networks (spatial variation) and the dynamic nature of context (temporal variation). We argue that these changes are significant making any manual approach to context adaptation very difficult.

1.1.2.1 Spatial Variation of Context

As network and security practitioners, it should come as no surprise that different networks exhibit different characteristics. What we have found during our research, however, is that these differences are surprisingly large, pervasive, and have significant impacts on all aspects of the security in an organization. For example, consider the issue of an organization's vulnerability profile.

Table 1.1 compares the vulnerable population in these of four networks in two ways: by the operating system and by the TCP port. Of the four production networks (A/16, B/16, C/17 and D/19), two of these networks (A/16 and B/16) are academic networks and two (C/17 and D/19) are web server farms. The second largest operating system in network A/16 is surprisingly Cisco IOS, which is found in the wireless access points and routers in the academic campus. On the other hand, Apple Mac OS is the dominant operating system in network B/16. As expected, the web-server farms were dominated by Windows servers. While SSH seems to be predominant service found in A/16 and B/16, HTTP, FTP and SMTP seems to be the dominant services in the web server farms. Therefore, the vulnerability profile may differ significantly depending on the network.

Today's attacks are global and everyone see the "same stuff" right? Unfortunately, the threat landscape also differs significantly for different networks. Cooke *et al.* [24] monitored unused address spaces (darknets) in different networks. Since unused addresses do not have any legitimate services, the traffic directed to these addresses are suspicious. Figure 1.1 shows the packet rates observed by different sensors and normalized by /24 address range. It shows that some networks receive significantly more suspicious traffic than others. In Bailey *et al.* [17], we examined, for 31 darknets, the top 10 destination ports, based on the number of packets, and compared these lists across darknets. Figure 1.1 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, a week, and a month time span. 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. Not only are there more or less attacks based on where you are, those attacks are targeting different services as well.

The traffic characteristics of a network may also be significantly different than others. For example, the list of IP addresses that legitimately access services on a given network may be different from other networks. Similarly, HTTP may be the most prominent protocol in web server farms and SMTP may be the most prominent protocol in a mail service provider network. Consider the data in Table 1.2. While some applications (e.g., web) are global popular, many are not (e.g., Lotus Notes, XBox). Note also the stark differences in the magnitude of traffic as well as the different behavior as either servers (e.g., high in bound traffic) or clients (i.e., high outbound traffic) of a particular

service.

1.1.2.2 Temporal Variation of Context

Another interesting observation of our work is that these unique individual contexts are highly dynamic in nature. Attack surfaces, usage models, and even vulnerability profiles change rapidly as new attacks are released, flash crowds are formed, or new application emerge. As an example, consider Table 3.2 which shows the top five TCP ports and the number of packets observed over a day for five months on the a /24 darknet in the B/16 network. We find that new services were targeted heavily each month. The TCP ports 6000 and 1080 were the unusual ones targeted in April, the TCP port 5000 was targeted in May, the TCP ports 22 and 5900 were targeted in June, and TCP port 4444 in July. The highly variable nature of this threat landscape makes chasing exploits difficult for the defenders, who must adjust their vision of the attack surface to today's or this week's most popular attacks.

1.2 Main Contributions

In the previous section, we found that the security context changes significantly with time and space. Our thesis is that such differences can be exploited to significantly improve the performance and accuracy of network-based security system. We justify our thesis by showing such improvement for three types of security systems namely: the known threat detection systems, the new threat detection systems and the reputation-based systems. We first explore how context information in the form of network traffic and signature set can be used to improve performance of signature-based security systems. We then look at why end-host characteristics and application configuration is important in the accuracy of the honeynet system and show how it can be automatically used to improve detection of new threats. Third, we investigate how network traffic impacts the accuracy of a reputation-based system like blacklist generation and show how traffic on a network may be utilized to generate accurate blacklists. We elaborate on these ideas in the remaining part of the thesis and we start by providing an overview of our main contributions:

- **Workload-aware intrusion detection**

Intrusion detection and prevention systems (IDS/IPS) take a set of signatures and detect intrusions by matching them with network traffic. Existing approaches to signature evaluation apply statically-defined optimizations that do not take into account the network in which the IDS or IPS is deployed or the characteristics of the signature database. We argue that for higher performance, IDS and IPS systems should adapt according to the workload, which includes the set of input signatures and the network traffic characteristics.

We developed an adaptive algorithm that systematically profiles attack signatures and network traffic to generate a high performance and memory-efficient packet inspection strategy. We implemented our idea by building two distinct components over Snort: a profiler that analyzes the input rules and the observed network traffic to produce a packet inspection strategy, and an evaluation engine that pre-processes rules according to the strategy and evaluates incoming packets to determine the set of applicable signatures. We have conducted an extensive evaluation of our workload-aware Snort implementation on a collection of publicly available datasets and on live traffic from a border router at a large university network. Our evaluation shows that the workload-aware implementation outperforms Snort in the number of packets processed per second by a factor of up to 1.6x for all Snort rules and 2.7x for web-based rules with reduction in memory requirements. Similar comparison with Bro shows that the workload-aware implementation outperforms Bro by more than six times in most cases.

- **Network-aware honeynet configuration**

Honeynet is a collection of sacrificial hosts explicitly deployed to be scanned, compromised, and used in attacks. Honeynets have recently become popular to detect and characterize threats such as worms, botnets and malware. Unfortunately, existing approaches to deploying honeynets largely ignore the problem of configuring operating systems and applications on individual hosts, leaving the user to configure them in a manual and often ad hoc fashion. We demonstrate that such ad hoc configurations are inadequate: they misrepresent the security landscape of the networks they are trying to protect and are relatively easy for attackers to discover. Therefore, a honeynet configuration should take the deployment context i.e., the network in which it is deployed to provide visibility into attacks and resistance to fingerprinting.

We show that manually building honeynet configurations for each network is hard, as each network has its own unique threat and vulnerability spaces, and the potential number of hosts to configure in the honeynet is quite large. We argue that honeynets with individually consistent hosts and proportional representation of the network will achieve the two desired goals of visibility into network attacks and resistance to discovery. We develop an automated technique based on profiling the network and random sampling to generate these honeynet configurations. Through experimental evaluation and deployment of configurations generated by our technique, we demonstrate significantly more visibility and higher resistance to discovery than current methods.

- **Context-aware blacklist generation system**

Blacklists have become popular among the operational community to filter or block the explosive growth of unwanted traffic on the Internet. Blacklists generated from firewall logs are used to block compromised hosts and blacklists generated from spamtraps are used to block spam. While these techniques have gained prominence, little is known about their effectiveness and potential draw backs.

We performed a preliminary study on the effectiveness of reputation-based blacklists namely those that are used for spam detection. We examined the effectiveness, in terms of false positives and negatives, of four blacklists, namely NJABL, SORBS, SpamHaus and SpamCop and investigated into the sources of the reported inaccuracy. We found that the black lists studied in our network exhibited a large false negative rate. NJABL had a false negative rate of 98%, SORBS had 65%, SpamCop had 35% and SpamHaus had roughly 36%. The false positive rate of all blacklists were low except that of SORBS, which had an overall false positive rate of 10%. The false positive of SORBS came mostly from blacklisting six Google mail servers that sent significant amount of ham to our network. However, since very little is known about the approaches taken by these services to generate their blacklists, and only the results of the generation are available (not the raw data), no one has explored in depth the reasons for these failures.

To solve this problem, we propose a new context-aware approach to blacklist generation. By making use of local usage and reachability information, as well as the global information

provided by blacklists, we can provide a significant improvement over existing approaches. In particular, this context-aware paradigm enables two specific techniques: ratio-based blacklisting and speculative aggregation. In the ratio-based blacklisting approach, the traffic on the live network is compared to the traffic on the spamtraps to determine if it is safe to blacklist an IP address. We call this approach the ratio-based approach as the ratio of email messages on the live network to the email messages on the spamtrap is used as a measure to blacklist an IP address. In the second approach, speculative aggregation, we use local reachability information as well as application history to predict where new spam messages will come while limiting the chance that these predicted hosts or networks are of use to the local network. A deployment of context-aware blacklists for over a month in a large academic network demonstrated significant improvement in blacklist accuracy.

1.3 Organization of the Dissertation

The remaining dissertation is structured as follows: Chapter 2 shows how performance of an intrusion detection and prevention system can be significantly improved by adapting to the network traffic and the rule set. Chapter 3 argues that configuration of a honeynet should account for surrounding network and proposes an automated approach to configure such honeynets. Chapter 4 shows that the current methods for generating blacklists are inaccurate and presents a more accurate blacklist generation approach that is aware of the network context. Finally, Chapter 5 concludes our current work and presents future plans to explore new types of context information to improve network-based security systems.

CHAPTER 2

Workload Aware Intrusion Detection

Intrusion Detection Systems (IDS) and Intrusion Prevention Systems (IPS) are widely deployed in a large number of organizations to detect and prevent attacks against networked devices. This chapter explores the performance bottlenecks of IDSs and IPSs and demonstrates that an adaptation to the deployment context will significantly speed up these systems.

The core component of popular IDSs, like Snort [67], is a deep packet inspection engine that checks incoming packets against a database of known signatures (also called rules). The performance of this signature-matching system is critical to the scalability of IDS and IPS systems, including packet per second rate. The dominant factor in determining the performance of this signature matching engine, whether implemented in software or hardware, is the number and complexity of the signatures that must be tested against incoming packets. However, both the number and complexity of rules appears to be increasing. For example, the recent Windows Meta-File (WMF) exploit [49] required inspecting and decoding more than 300 bytes into the HTTP payload which could quickly overwhelm the CPU of the IDS or IPS, causing massive packet drops [43].

As a result, there has been significant effort in developing methods for efficient deep packet inspection. Current IDSs like Snort and Bro attempt to evaluate as few rules as possible in a highly parallel way. For example, Snort pre-processes rules to separate them by TCP ports, and then parallelizes the evaluation based on port. However, these groupings can be inefficient because all of the rules in a given group do not apply to incoming packets. Moreover, separating rules by multiple protocol fields in a naive way does not solve the problem because of the additional memory overhead associated with managing groups.

In this chapter, we argue that IDS and IPS should dynamically adapt the parallelization and separation of rules based on the observed traffic on the network and the input rules database. That is, all IDS and IPS workloads are not the same, and systems should adapt to the environment in which they are placed to effectively trade-off memory requirements for run-time rule evaluation. To demonstrate this idea, we have developed an adaptive algorithm that systematically profiles the traffic and the input rules to determine a high performance and memory efficient packet inspection strategy that matches the workload. To effectively use memory for high performance, the rules are separated into groups by values of protocol fields and then these rule groups are chosen to be maintained in memory following a simple idea of "the rule groups that have a large number of rules and match the network traffic only a few times should be separated from others." This idea follows our observation that if rules with value v for a protocol field are grouped separately from others, then for any packet that does not have value v for the protocol field, we can quickly reject all those rules, and if only a few packets have that value, then those rules will be rejected most of the time. Therefore, our workload-aware scheme aims to determine a small number of effective groups for a given workload.

Our algorithm determines which rule groups are maintained in the memory by choosing protocol fields and values recursively. It first determines the protocol field that is most effective in rejecting the rules, and then separates those groups with values of the chosen protocol field that reject at least a threshold number of rules. After forming groups for each of these values, the algorithm recursively splits the groups by other protocol fields, producing smaller groups. In this way, we generate a hierarchy of protocol fields and values for which groups are maintained. By lowering the threshold, memory can be traded-off for performance. Using this systematic approach for computing a protocol evaluation structure, we automatically adapt an IDS for a given workload.

In this chapter we develop a prototype Snort implementation based on our workload-aware framework, which we call Wind. The implementation has two main components. The first component profiles the workload (i.e., the input rules and the observed network traffic) to generate the hierarchical evaluation tree. The second component takes the evaluation tree, pre-processes the rules, and matches incoming packet to the rules organized in the tree.

We evaluate our prototype workload-aware Snort implementation on the widely recognized DARPA intrusion detection datasets, and on live traffic from a border router at a large live aca-

demic network. We find that our workload-aware algorithm improves the performance of Snort up to 1.6 times on all Snort rules and up to 2.7 times for web-based rules. Surprisingly, we also find that the algorithm reduces memory consumption by $10-20\%$. We also compare the workload-aware algorithm with Bro, and find it outperforms Bro by more than six times on most workloads.

To summarize, the main contributions of this chapter are:

- We propose a method for improving the performance of IDS and IPS systems by adapting to the input rules and the observed network traffic.
- To demonstrate our idea, we constructed a workload-aware Snort prototype called Wind that consists of two components: a component that profiles both the input rules and the observed network traffic to produce an evaluation strategy, and a second component that pre-process the rules according to the evaluation strategy, and then matches incoming packets.
- We evaluate our prototype on publicly-available datasets and on live traffic from a border router. Our evaluation shows that Wind outperforms Snort up to 1.6 times and Bro by six times with less memory requirements.

The rest of the chapter is organized as follows: Section 2.1 discusses the related work. Section 2.2 presents the design of Wind and Section 2.3 presents empirical results comparing Wind with existing IDSs. Section 2.4 discusses techniques for dynamically adapting Wind to changing workloads. We finally conclude with directions for future work in Section 2.5.

2.1 Related Work

The interaction between high-volume traffic, number of rules, and the complexity of rules has created problems for Intrusion Detection Systems that examine individual flows. Dreger *et. al.* [29] present practical problems when Intrusion Detection Systems are deployed in high-speed networks. They show that current systems, like Bro [60] and Snort [67], quickly overload CPU and exhaust the memory when deployed in high-volume networks. This causes IDS to drop excessive number of packets, some of which may be attack incidents. Therefore, they propose some optimizations to reduce memory consumption and CPU usage which are orthogonal our approach.

Lee *et. al.* [46] find that it is difficult to apply all possible rules on an incoming packet. Therefore, they evaluated the cost-benefit for the application of various rules and determined the best set of rules that can be applied without dropping packets. However, they trade-off accuracy for achieving high bandwidth. Kruegel and Valeur [45] propose to slice traffic across a number of intrusion detection (ID) sensors. The design of their traffic slicer ensures that an ID sensor configured to apply certain rules on a packet does not miss any attack packet.

Sekar *et. al.* [69] developed a high-performance IDS with language support that helps users easily write intrusion specifications. To specify attack signatures within a payload, they used regular expressions. This specification is different from Snort in which attack signatures contain exact substrings, in addition to regular expressions, to be matched with a payload. Using regular expressions is a more generic approach than using substrings to specify an attack signature. However, regular expressions are more expensive to evaluate than exact substring matches. (The complexity of checking a regular expression of size m over a payload of size n is $O(mn)$ [40] and it is more expensive than checking for exact substring within a payload, which has a time complexity of $O(n)$ [40]). Aho-Corasick [40] matches a *set* of substrings over a payload in $O(n)$. Alternative schemes like Wu-Manber [86] speed up matching by processing the common case quickly. The multi-pattern optimizations to speed up an Intrusion Detection System are complementary to our approach, as we speed up an IDS by reducing the expected number of patterns to be checked with a packet.

Versions of Snort prior to 2.0 evaluated rules one by one on a packet. This required multiple passes of a packet and the complexity of intrusion detection grew with the number of rules. To eliminate redundant checking of protocol fields, rules that have the same values for a protocol field can be pre-processed and aggregated together. Then, a check on the protocol field value would equivalently check a number of rules. By clustering rules in this way and arranging the protocol fields by their entropy in a decision tree, Kruegel and Toth [44], and Egorov and Savchuk [30] independently demonstrated that Snort (version 1.8.7) performance can be improved up to three times. However, these papers only examined the input rules to determine the rule evaluation order. In contrast, we analyze the traffic, as well as the rules, to determine the rule evaluation order. Secondly, they use entropy as an ordering metric, whereas we use a more intuitive metric for selecting as few rules as possible. Lastly, a naive arrangement of protocol fields would drastically increase memory usage, and these papers have not considered the memory costs associated with their approaches. Wind

improves performance and at the same time reduces the memory usage of an intrusion detection and prevention system.

Snort 2.0 [56] uses a method in which rules are partitioned by TCP ports, and a packet's destination and source port determines the sets of applicable rules. Then, the content specified by these applicable rules are checked in one pass of the payload, using either the Aho-Corasick or the Wu-Manber algorithm, for multiple substring search. If a substring specified in some attack rule matches with the packet, then that rule is evaluated alone. We found that the parallel evaluation significantly sped up Snort. Snort now takes 2-3 microseconds per packet, when compared to earlier findings of 20-25 [30] microseconds per packet for Snort versions prior to 2.0 [1]. This optimization significantly improved Snort performance. Nevertheless, we further speed up a multi-rule Snort on many workloads. This is achieved by leveraging the workload to partition rules in an optimized evaluation structure.

Recently, specialized hardware [68, 21] for intrusion detection in high-volume networks has been developed. These solutions have used Field-programmable gate array (FPGA) to implement the intrusion detection systems. As a result hardware-based solutions are complex to modify (e.g., to change the detection algorithm). The automated adaptation technique presented in this paper can be implemented using FPGA's as well and further improve the performance of these systems.

Our work is also related, and inspired, by database multi-query optimization methods that have long been of interest to the database community (see [32, 71, 70, 59] for a partial list of related work). However, rather than finding common subexpressions amongst multiple SQL queries against a static database instance, the problem that we tackle requires designing a hierarchical data structure to group network rules based on common subexpressions, and using this data structure in a data streaming environment.

2.2 Designing a Workload-Aware IDS

In this section, we first show that checking a protocol field can reject a large number of rules, and the number of rejected rules varies significantly with the protocol field. Then, we take this observation a step further and construct an evaluation strategy that decomposes the set of rules recursively by protocol fields and constructs a hierarchical evaluation tree. However, a naive strategy

[1] The difference in computing systems and rules are not taken into account for a rough discussion.

that separates rules by all values of a protocol field will use too much memory. To address this issue, we present a mathematical model that addresses the trade-off between memory occupied by a group of rules and the improvement in run-time packet processing. Finally, we present a novel algorithm and a concrete implementation to capture statistical properties of the traffic and the rule set to determine a high-performance and memory-efficient packet inspection strategy.

2.2.1 Separating Rules by Protocol Fields

An IDS has to match a large number of rules with each incoming packet. Snort 2.1.3 [67] is distributed with a set of 2,059 attack rules. A rule may contain specific values for protocol fields and a string matching predicate over the rest of the packet. For example, a Snort rule that detects the Nimda exploit is shown below:

```
alert tcp EXTERNAL_NET any -> HOME_NET 139 (msg:``NETBIOS nimda .nws'';
                content:``|00|.|00|N|00|W|00|S";)
```

This rule matches a packet if the value of transport protocol field is TCP, the value in the source address field matches the external network, the destination address field contains an address in the home network, the value of destination TCP port field is 139, and if the payload contains the string ``|00|.|00|N|00|W|00|S".

A simple approach for evaluating multiple rules on an incoming packet is to check each rule, one-by-one. However, this solution involves multiple passes over each packet and is too costly to be deployed in a high-speed network. Therefore, the evaluation of the rules should be parallelized as much as possible and evaluated in only a few passes over the packet. To evaluate a protocol field in the packet only once, we need to pre-process rules and separate them by the values of the protocol field. Then, by checking the value of just one protocol field, the applicable rules can be selected. The advantage of separating rules by the protocol field values is that a large number of rules can be rejected in a single check. In Snort, the rules are pre-processed and grouped by destination port and source port. The TCP ports of an incoming packet are checked to determine the set of rules that must be considered further, and all other rules are immediately rejected. The expected number of rules that will be rejected by checking a protocol field of an incoming packet depends on two factors: the traffic characteristics and the rule characteristics. Consider an input rule set with a large number of rules that check if the destination port is 80. Assuming that the rules are grouped together

by the destination port, for a packet not destined to port 80, a *large* number of port-80 rules will be rejected immediately. If only a few packets are destined to port 80, then a *large* number of rules will be rejected *most* of the time.

Figure 2.1 shows the number of rules that can be rejected immediately for an incoming packet when rules are grouped by different protocol fields. For this figure, we used the $2,059$ rules that came with Snort 2.1.3 distribution and the traffic is from the Thursday on the fourth week of 99 DARPA dataset (99-test-w4-thu). Figure 2.2 shows a similar graph, using the same set of rules on a border router in a large academic network. The graphs show that checking the destination port rejects the maximum number of rules, which is followed by destination IP address and then by the check that determines whether the packet is from a client. The source IP address is fourth in the list for the border router traffic and seventh in the DARPA dataset. After this, most other protocol fields reject a small number of rules. Therefore, the graphs show that the rule set and the traffic mix cause varying number of rules to be rejected by different protocol fields.

Now, checking whether a payload contains a particular string is a costly operation, but checking the value of a protocol field is cheap. So, it is preferable to check protocol fields to reduce the number of applicable rules. To use multiple protocol fields for reducing the applicable rules, the rules have to be pre-processed in a hierarchical structure in which each internal node checks a protocol field and then divides the rules by the values of the protocol field. Finally, the leaf node is associated with a set of rules and a corresponding data structure for evaluating multiple patterns specified in the rules. We are agnostic to the multi-pattern search algorithm and the only objective of this hierarchical evaluation is to reduce the number of applicable rules, so that a packet is matched with as few rules as possible.

Figure 2.3 shows an example of an evaluation tree in which protocol fields are hierarchically evaluated to determine the set of applicable rules. It first checks for destination port. If the destination port matches a value for which the set of rules is maintained then those groups of rules are further analyzed, or else the generic set of rules is picked. If the destination port is 21, the connection table is checked to determine if the packet came from the client who initiated the connection, and the corresponding rules are picked. If the destination port is 80, then the destination IP address of the packet is checked. Then, depending on whether the packet is destined to the Home Network or not, the correct set of rules are picked to further evaluate on the packet. However, maintaining

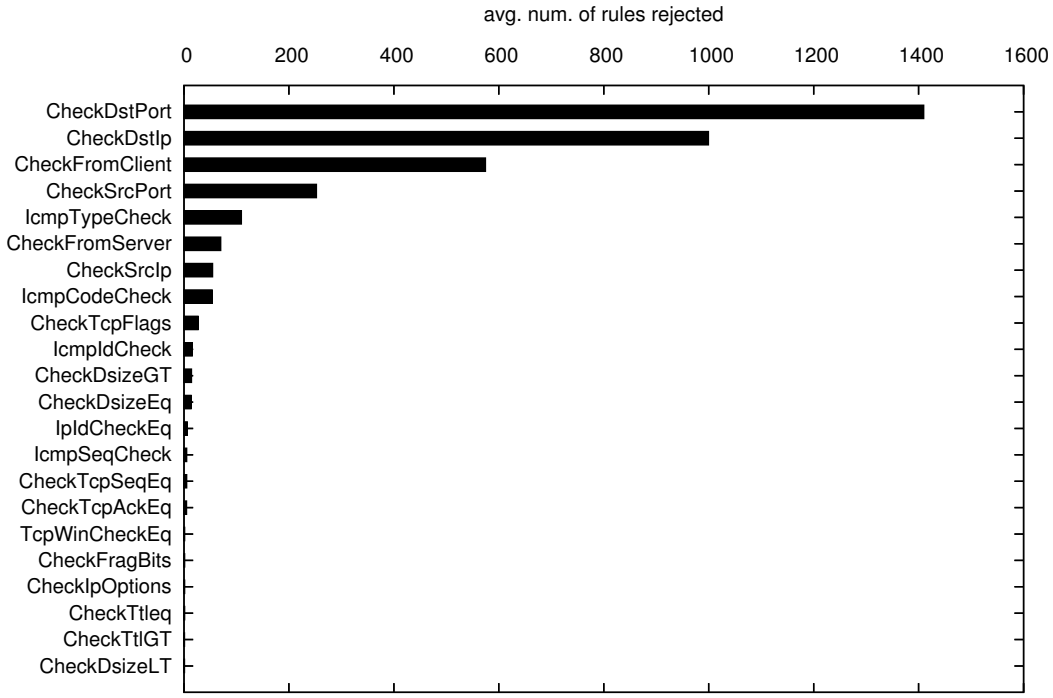


Figure 2.1: Average number of rules (out of $2,059$) rejected by checking different protocol fields for the DARPA dataset (99-test-w4-thu).

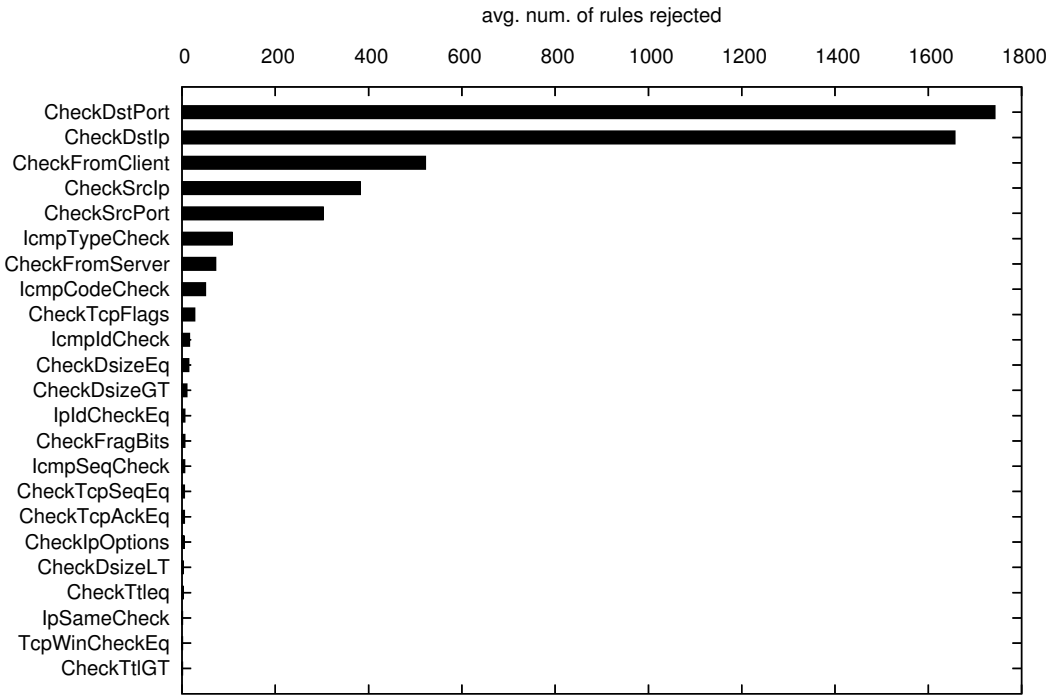


Figure 2.2: Average number of rules (out of $2,059$) rejected by checking different protocol fields for data from the border router of a large academic network.

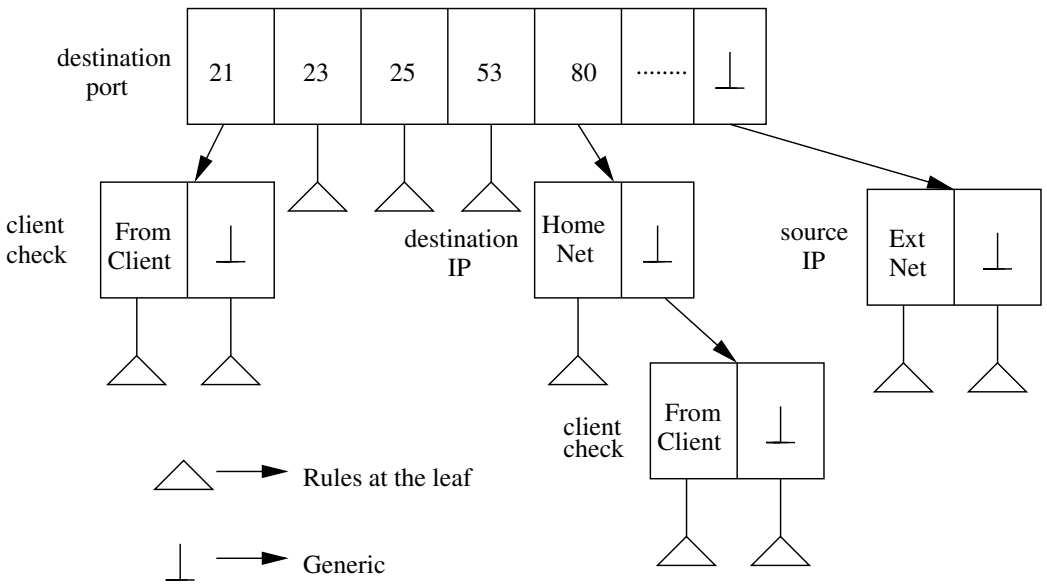


Figure 2.3: An example evaluation tree that checks protocol fields to determine the set of rules for matching an incoming packet.

a naive hierarchical index structure, in which every specific value of a protocol field is separated, consumes a significant amount of memory for the following two reasons:

1. **Groups require memory:** Multiple patterns from a set of rules have to be searched in a payload in only one pass of the payload. Therefore, additional data structures are maintained for fast multi-pattern matching. This structure can be a hash table as in the case of the Wu-Manber [86] algorithm, or a state table as in the case of the Aho-Corasick [40] algorithm. These structures consume a significant amount of memory.

2. **Rules are duplicated across groups:** If groups are formed by composing two protocol fields hierarchically, then the number of distinct groups may increase significantly. For example, assume that the rules are first divided by destination port, and then each group so formed is further divided by source port. A rule that is specific in source port but matches *any* destination port has to be included in all groups with a particular destination port. If the groups that are separated by destination port are further divided by source port, then separate source port groups would be created within all the destination port groups. For a set of rules with n source port groups and m destination port groups, the worst number of groups formed, when rules are hierarchically arranged by the two protocol fields, is $n \times m$.

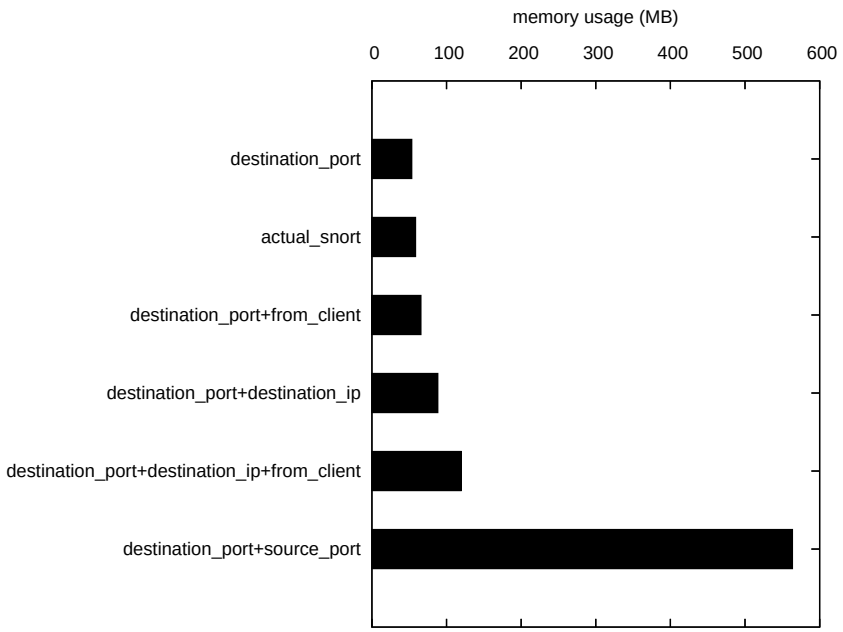


Figure 2.4: Memory usage when rules are hierarchically arranged by protocol fields in the specified order using the DARPA dataset (99-w4-thu).

To investigate the memory consumed when rules are grouped hierarchically by different protocol fields, we instrumented Snort to construct this structure for a given list of protocol fields. We then measured the memory consumed for different combinations of protocol fields. Figure 2.4 shows the memory consumed when different protocol fields are hierarchically arranged, and a separate bin is maintained for every specific value in a protocol field (trace data was 99-w4-thu from DARPA dataset and the 2059 rules of Snort-2.1.3 distribution). This shows that the memory consumed by the combination of destination port and client check is 50% more than just the destination port. The memory required for the combination of destination port and destination IP address is two times, and for the combinations of destination port, destination IP address and client check, the memory consumed is three times than only using the base destination port. From the graph, the increase in memory is evident when the rules are hierarchically grouped by destination port and source port. Therefore, constructing such a hierarchy immediately raises two important questions:

1. What is the order in which the protocol fields are evaluated in the hierarchy?

2. What are the values of a protocol field for which groups are maintained?

In what follows, we first present a mathematical description of this problem, analyzing the cost and benefit of different orders of protocol fields and the field values for which the rules are maintained. We then argue that these questions can be answered by capturing properties of the workload, namely the traffic-mix characteristics and the input rule set characteristics.

2.2.2 Formal Description

In this section, we formulate the problem of determining the order of evaluation and the values of protocol fields for which the groups are maintained. As argued earlier, the cost in maintaining a separate group is mostly the memory consumed by the group. Intuitively, the benefit obtained by maintaining a group of rules can be measured by how *many* rules this group separates from the rule set and how *frequently* this group is rejected for an incoming packet. We begin by formalizing the problem.

Consider n protocol fields $F_1, F_2, \ldots, F_n$. Let $v_1^i, v_2^i, \ldots, v_{m_i}^i$ be m_i specific values of the protocol field F_i present in various rules in the rule set. Let $\mathcal{P} = (F_{r_1} = v_{j1}^{r_1}) \wedge (F_{r_2} = v_{j2}^{r_2}) \wedge \ldots \wedge (F_{r_i} = v_{ji}^{r_i})$ be the predicate for a group that is picked only when the packet matches specific values for i protocol fields, and $f(\mathcal{P})$ denote the probability that the protocol fields for an incoming packet matches the predicate $\mathcal{P}$, i.e., $v_{j_1}^{r_1}$ for protocol field F_{r_1}, $v_{j_2}^{r_2}$ for protocol field F_{r_2}, ..., $v_{j_i}^{r_i}$ for protocol field F_{r_i}. $f(\mathcal{P})$ actually captures statistics on the network traffic. The probability that an incoming packet does not have the values for protocol fields as the predicate $\mathcal{P}$ is $1 - f(\mathcal{P})$. Let the benefit of rejecting rule R be measured by improvement of b_R in run time. Then, every time a packet does not have the values for protocol fields as $\mathcal{P}$, benefit of $\sum_{R=rule\ with\ value\ \mathcal{P}} b_R$ is obtained by maintaining a separate group of rules with values $\mathcal{P}$. Therefore, the overall benefit of creating a group with specific values for protocol fields present in the predicate $\mathcal{P}$ includes traffic characteristics in $f(\mathcal{P})$ and rule properties in the rule set as:

$$(1 - f(\mathcal{P})) \times \sum_{R=rule\ with\ value\ \mathcal{P}} b_R \tag{2.1}$$

Assume that $c(\mathcal{P})$ is the memory cost of creating a group for a set of rules that satisfies the

predicate $\mathcal{P}$. Then, the problem of an effective hierarchical structure is to determine the set of groups such that they maximize the benefit measured by improvement in run time for a given total cost, measured by the total amount of memory that is available. Formally, the objective is to determine m and m distinct predicates $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_m$ that maximizes

$$\sum_{i=1}^{m} [(1 - f(\mathcal{P}_i)) \times \sum_{R=rule\ with\ value\ \mathcal{P}_i} b_R] \tag{2.2}$$

with the cost constraint

$$\sum_{i=1}^{m} c(\mathcal{P}_i) \leq maximum\ memory \tag{2.3}$$

The problem is similar to minimizing the number of leaf nodes in a decision tree [31]. However, decision tree problem assumes that there is no ambiguity in element classification and that there is no overlap between the groups. In our problem, a rule can be possibly placed in multiple groups and the objective is to identify the set of groups that maximizes the benefit function.

2.2.3 Our Approach

In this section, we design an algorithm that captures the properties of input rules and traffic characteristics to produce an effective set of rule groups, separated by values of protocol fields. These groups are then arranged in a hierarchical evaluation structure, which determines the order in which protocol fields are evaluated on an incoming packet. We begin with some assumptions that simplify the above mathematical model for a realistic treatment and then present our algorithm.

2.2.3.1 Assumptions

It is not easy to precisely determine the cost of creating a data structure for matching multiple patterns and the absolute benefit achieved by rejecting a rule. For some exact substring-match algorithms (like Aho-Corasick), the memory space occupied by the data structure may not grow linearly with the number of patterns. For hash-based algorithms, the memory consumed is independent of the number of patterns. This makes estimating cost for a multi-pattern-matching algorithm difficult. At the same time, for most algorithms that perform multi-pattern matching, it is hard to estimate the benefit of excluding a single pattern. Therefore, we will make two simplifying assumptions that allows us to easily compute the cost and benefit:

1. The cost of creating a multi-pattern data structure for any group of patterns is constant. This assumption is valid for hash-based matching algorithms, like Wu-Manber, that allocate fixed hash space. However, this assumption is incorrect for the Aho-Corasick algorithm in which the required space may increase with the increase in the number of patterns.

2. The benefit of rejecting any rule is a one-unit improvement in run time (i.e., $b_R = 1$) except for rules that have content of maximum length one. The rules that have content length one significantly degrade multi-pattern matching and should be separated if possible. Therefore, rules with a content length of one are assigned a large benefit (mathematically infinity). It is possible that other patterns may adversely impact the performance in multi-pattern search, but we choose to ignore such interactions for simplicity.

It is important to note that our assumptions help us to easily estimate the cost and benefit of creating a group, and more accurate estimates will only improve our scheme.

2.2.3.2 The Algorithm

Instead of specifying a fixed memory cost and then maximizing the benefit, we specify the trade-off between the cost and the benefit. We say that any specific value of a protocol field that rejects at least a minimum THRESHOLD number of rules should be assigned a separate group and hence, memory space. This specification allows us to more easily tune real-time performance.

The mathematical model allows us to compare two groups with specific values for a number of protocol fields. The problem is then to determine a set of groups in which each group rejects at least a THRESHOLD number of rules and the set maximizes the overall benefit. However, this may require generating all possible sets, which is computationally infeasible. Therefore, we do not attempt to produce an optimal set of groups, but instead to discover possible groups heuristically. The main intuition behind our algorithm is to place all rules in a bin and iteratively split that bin by the protocol field that produces the maximum benefit, and at each split separate values of the chosen protocol field that reject at least a THRESHOLD number of rules on average.

We now explain our algorithm in detail. First, all rules are placed in a bin. Then, a few packets are read from the network and protocol fields in each rule are evaluated. Then, the benefit obtained by a value in a protocol field is computed using the benefit Equation 2.1. For value v_j^i of the protocol

field F_i, $f(\mathcal{P})$ reduces to $f(F_i = v_j^i)$ and $\sum_{R=rule\ with\ value\ \mathcal{P}} b_R$ reduces to $S_{F_i=v_j^i}$ where $S_{\mathcal{P}}$ indicates the number of rules with protocol field values specified by the predicate $\mathcal{P}$. This simplification is possible because b_R is one. The overall benefit of a protocol field is the sum of benefit of all values, and the protocol field is chosen that produces maximum benefit. Then groups are formed for each specific value in the protocol field that rejects at least THRESHOLD number of rules, or has a rule with content length one. Then, we partition the bin into those specific values and recursively compute other protocol fields for each of these bins. We stop splitting a bin if none of the protocol fields can reject at least THRESHOLD number of rules.

When we partition a bin into specific values, we replicate a rule that may match multiple of these specific values in all those bins. For example, if the rules are divided by destination port, then a rule that matches `'any'` destination port is included in all of those bins. This ensures that when a set of rules with a specific value for a protocol field are picked, other applicable rules are also matched with the packet. This is essential for correctness. Generally a rule with value v_j for a protocol field is included in a rule set with specific value v_i if $v_j \cap v_i \neq 0$. If there is an order in which the values are checked during run-time, then a rule v_j is included in v_i only if it appears before it, and if it satisfies the previous property.

Packets rejected by a protocol field may correlate with packets rejected by another protocol field, and so computing protocol fields independently may give misleading information. For example, a source port and a source IP address may reject exactly the same packets, in which case we do not gain anything by checking both of them. Our recursive splitting of a bin removes this problem of correlated values. This is because for a bin, we evaluate the benefit of remaining protocol fields only on those packets that match the values specified in the bin. For example, to split a bin containing port-80 rules, we only evaluate the remaining protocol fields on packets that have port 80. This ensures that the remaining protocol fields reject only the rules that were not rejected by port 80.

By choosing the protocol field that produces maximum benefit for each bin, we get an order in which the protocol field is checked for a packet. By choosing values that produce benefit above a threshold, we get the values that determines which groups should be maintained.

2.2.3.3 Implementation

We implemented two distinct components to develop a workload-aware Intrusion Detection System. The first component profiles the workload (i.e., the input rules and the live traffic) to generate the evaluation tree. The second component takes the evaluation tree, pre-processes the rules, and matches any incoming packet on the tree. These components are general enough to be applied to any IDS. We implemented our algorithm that generates an evaluation tree for a given workload over Snort 2.1.3. We chose Snort as it already provides an interface to read the rules into proper data structures. It also provides an interface to read the incoming traffic and check for different protocol fields.

As a second component, we modified Snort 2.1.3 to take the bin profiles and construct a hierarchical evaluation plan. Snort 2.0 [56] introduced an interface for parallel evaluation of rules on a packet. Our hierarchical evaluation tree provides the set of applicable rules for a packet according to its values for different protocol fields. We pre-computed the data structure required for parallel matching for each of these groups. For every packet, we used our evaluation tree to determine the set of applicable rules and allowed Snort to perform the evaluation. We implemented three protocol fields by which the hierarchical structure can be constructed, namely: destination port, source port, destination IP address, and whether the packet is from the client. Since rules contain a large number of distinct protocol fields and we want to immediately detect the applicable rules, we implemented a check for destination port using an array of 65,536 pointers. Source port and destination IP address was checked by looking for possible match in a linked list. We did this because only a few destination IP addresses/source ports have to be checked, and because maintaining a pointer for each specific value consumes significant memory. For client checks the rules were divided into two parts: those that required to check if the packet is coming from client, and the rest were others. Every time a bin was split, we ensured that a rule was included in all new bins whose specific value can match the value in the rule. This ensured the correctness of our approach. We also validated our system by matching the number of alerts that our system raises, when compared to the number of alerts raised by unmodified Snort on a large number of datasets.

2.3 Evaluation

In this section, we evaluate Wind on a number of publicly-available datasets and on traffic from a border router at a large academic network. On these datasets, we compared real-time performance of Wind with existing IDSs using two important metrics: the number of packets processed per second and the amount of memory consumed. To measure the number of packets processed per second, we compiled our system and the unmodified Snort with gprof [39] options and then evaluated the dataset with each one of them. Then we generated the call graph, using gprof, and examined the overall time taken in the `Detect` function, which is the starting point of rule application in Snort. Finally, using the time spent in `Detect` and the number of times it was called, we computed the number of packets processed per second. To compute the memory used, we measured the maximum virtual memory consumed during the process execution by polling each second the process status and capturing the virtual memory size of the process. We now describe the datasets and the computing systems that we used for our experiments.

2.3.1 Datasets and Computing Systems

We evaluated the performance of our system on a number of publicly-available datasets and on traffic from a large academic network. For publicly available datasets, we used traces that DARPA and MIT Lincoln Laboratory have used for testing and evaluating IDSs. We used two-week testing traces from 1998 [47], and two-week testing traces from 1999 [50]. This gave us 20 different datasets with home network 172.16.0.0/12. For evaluating the system on real-world, live traffic, we chose a gateway router to a large academic network with address 141.212.0.0/16. This router copies traffic from all ports to a span port, which can be connected to a separate machine for analyzing the traffic.

For DARPA dataset experiments, we used a dual 3.06 GHz Intel Xeon machine with 2 GB of main memory. The machine was running FreeBSD 6.1 with SMP enabled. We connected the span port of the gateway router to a machine with dual 3.0 GHz Intel Xeon processors and 2GB of main memory. The machine was running FreeBSD 5.4 with SMP enabled. The results that follow are the averages over 5 runs and with the THRESHOLD value set to 5.

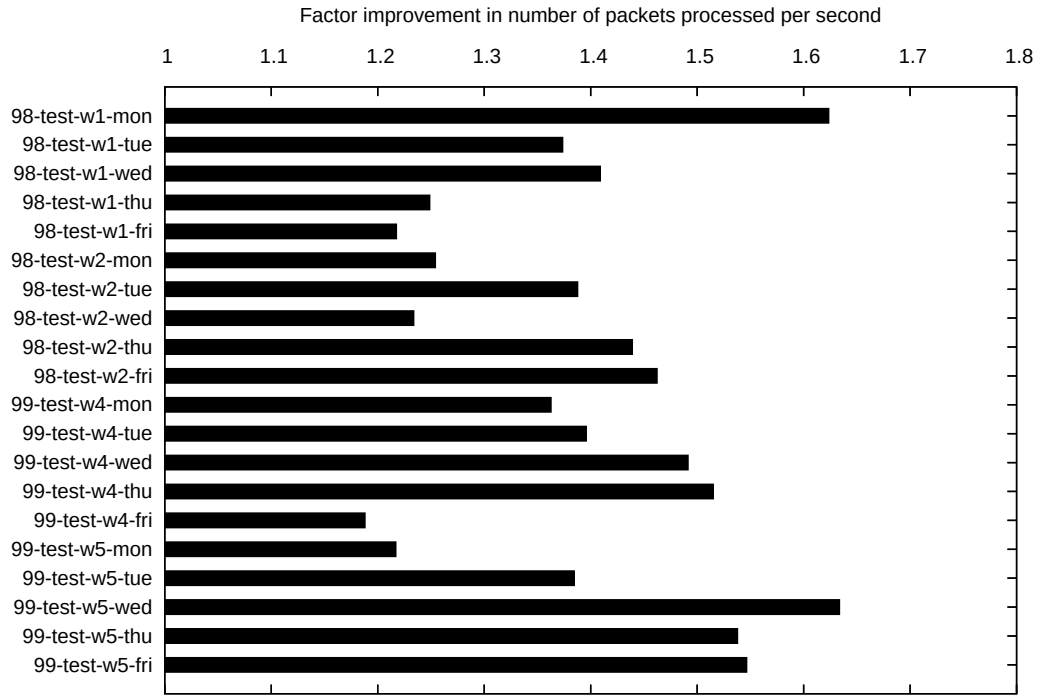


Figure 2.5: Factor improvement, in terms of number of packets processed per second, when compared to Snort for the 1998 and 1999 DARPA testing datasets.

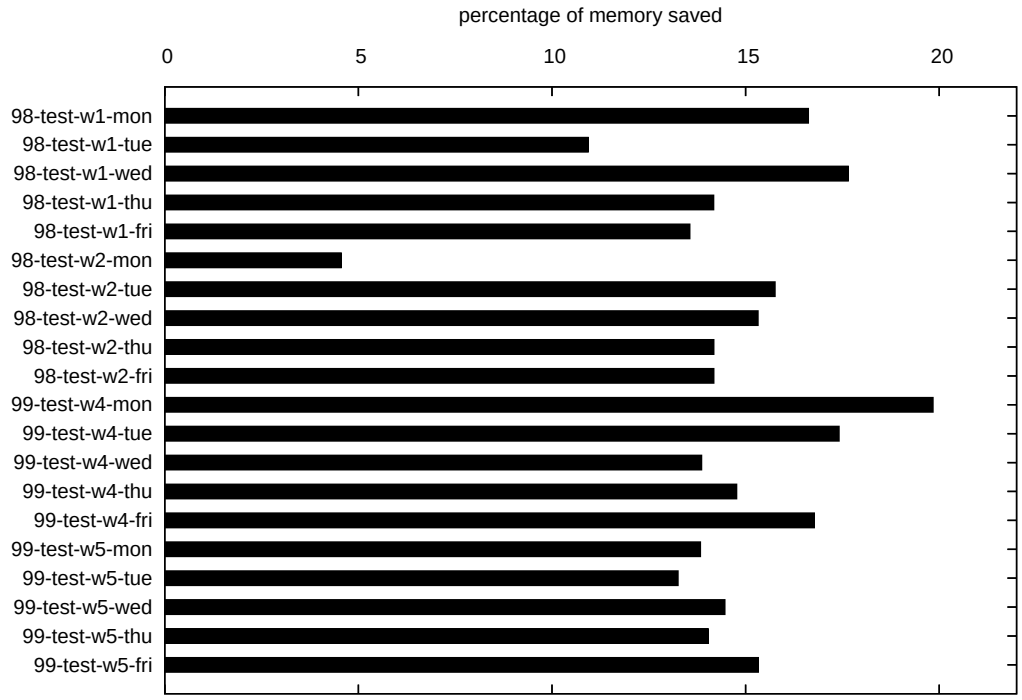


Figure 2.6: Percentage of memory saved for each of the 1998 and 1999 DARPA datasets, when compared to Snort.

2.3.2 Processing Time and Memory Usage

We compared Wind with Snort 2.1.3 for all rules included with the distribution. There were 2,059 different rules, and both Wind and Snort were run using default configuration. Figure 2.5 shows the amount by which we improved the number of packets processed per second by Snort. For most datasets, we find that our system processes up to 1.6 times as many packets as Snort. We also compared the memory used by our system with that of Snort. Figure 2.6 shows the memory saved by our system when compared to Snort. We find that our system uses about 10-20% less memory when compared to the unmodified Snort. In other words, we perform up to 1.6 times better in processing time and save 10-20% of the memory.

Wind and Snort were run on the border router for analyzing a million packets at a few discrete times in the week. Figure 2.7 shows the amount by which Wind improved the number of packets processed per second by Snort. It shows that the improvement factor on this dataset varied from 1.35 to 1.65. During the runs, Wind consumed 10-15% less memory than Snort.

2.3.3 Application-specific Rules

Until now, all our experiments were conducted by enabling all rules that came with the Snort distribution. However, in many networks, only application-specific rules can be used. For example, in many enterprise networks, the only open access through the firewall is web traffic. Since web traffic forms the dominant application allowed in many networks, we compared our system with Snort for web-based rules [2]. Figure 2.8 shows the magnitude by which our system improves Snort, in the terms of number of packets processed per second, for traffic at the border router. We found that for web-based rules, our system improves performance by more than two times when compared to Snort. Figure 2.9 shows a similar graph for the DARPA datasets. We observed that Wind outperforms Snort by a factor of up to 2.7 times. In this case, we saved 2-7% of the memory when compared to Snort.

[2] web-cgi, web-coldfusion, web-iis, web-frontpage, web-misc, web-client, web-php, and web-attack rules with Snort 2.1.3

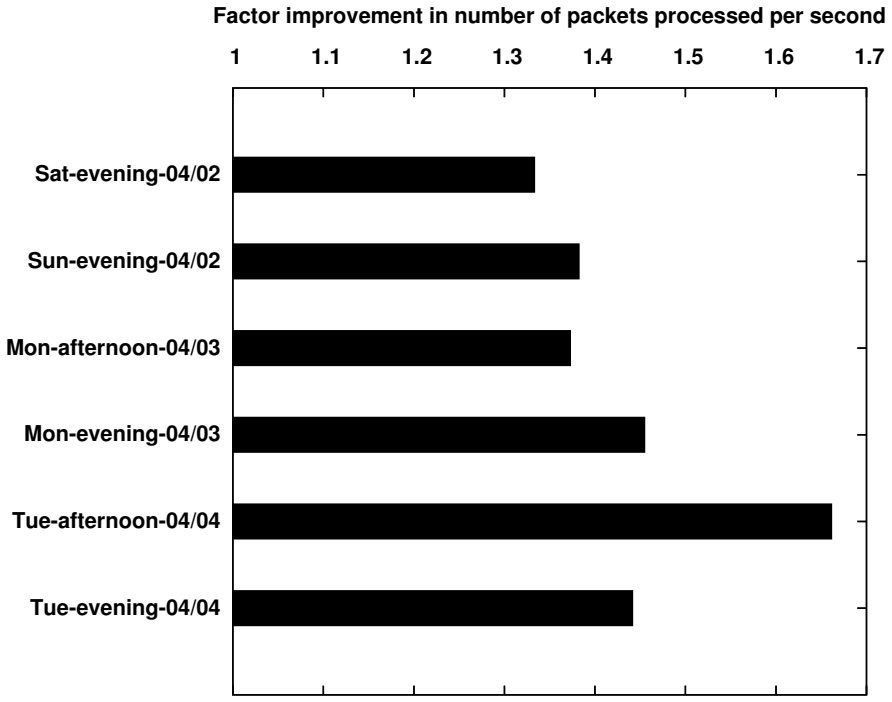


Figure 2.7: Factor improvement in number of packets processed per second, when compared to Snort, on data from a border router in an academic network.

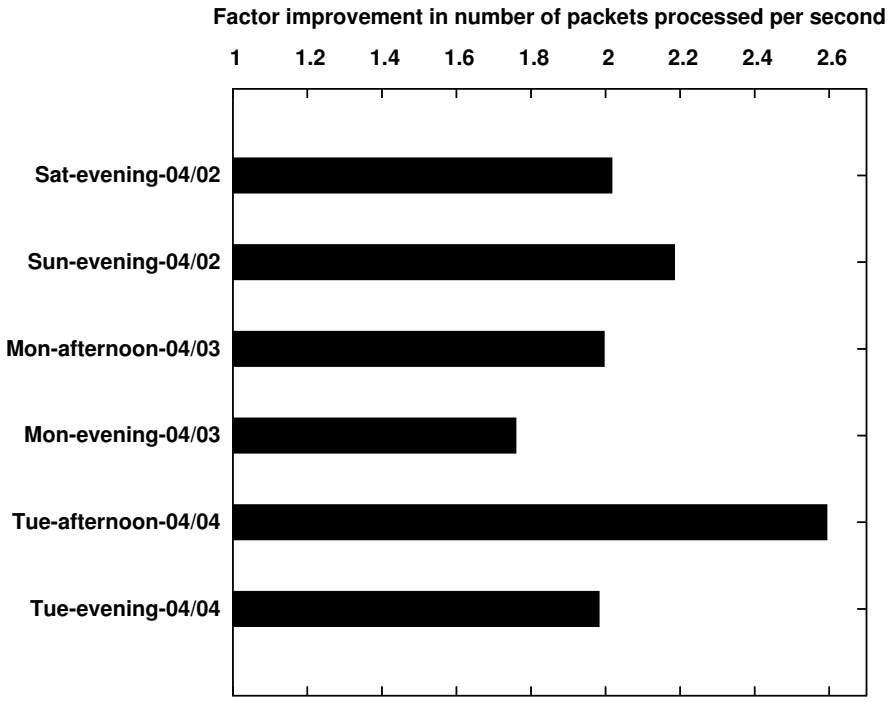


Figure 2.8: Factor improvement in number of packets processed per second, when compared to Snort, for web-based rules. These experiments were on traffic from a border router at an academic network.

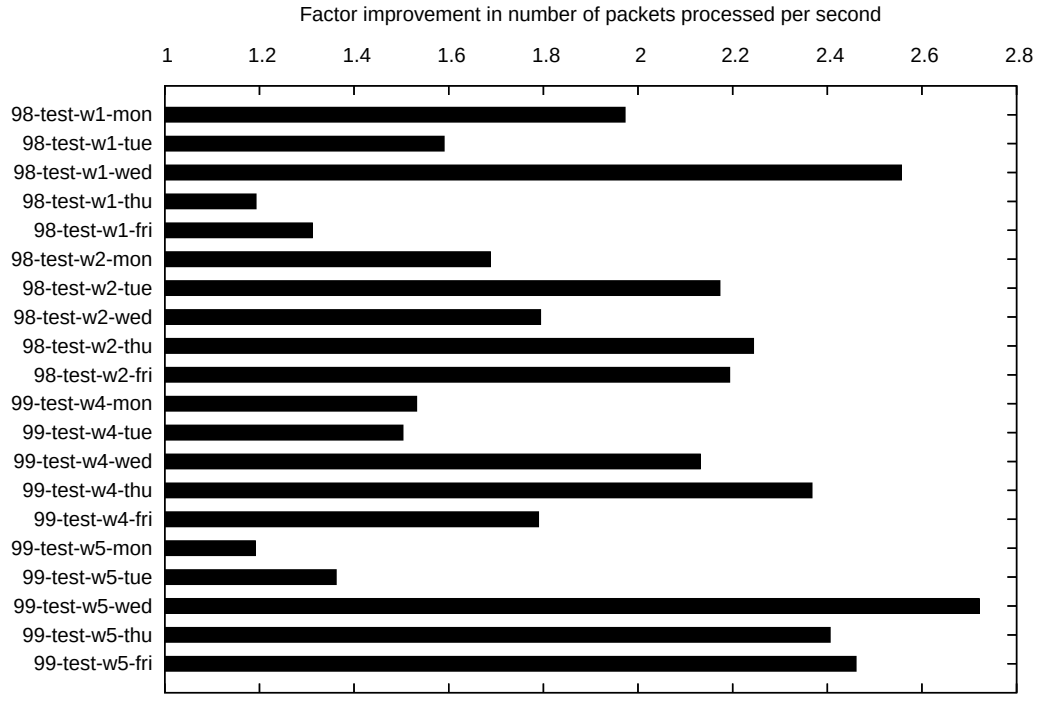


Figure 2.9: Factor improvement in number of packets processed per second by Wind when compared to Snort for web-based rules. The datasets include the 1998 and 1999 DARPA intrusion detection datasets.

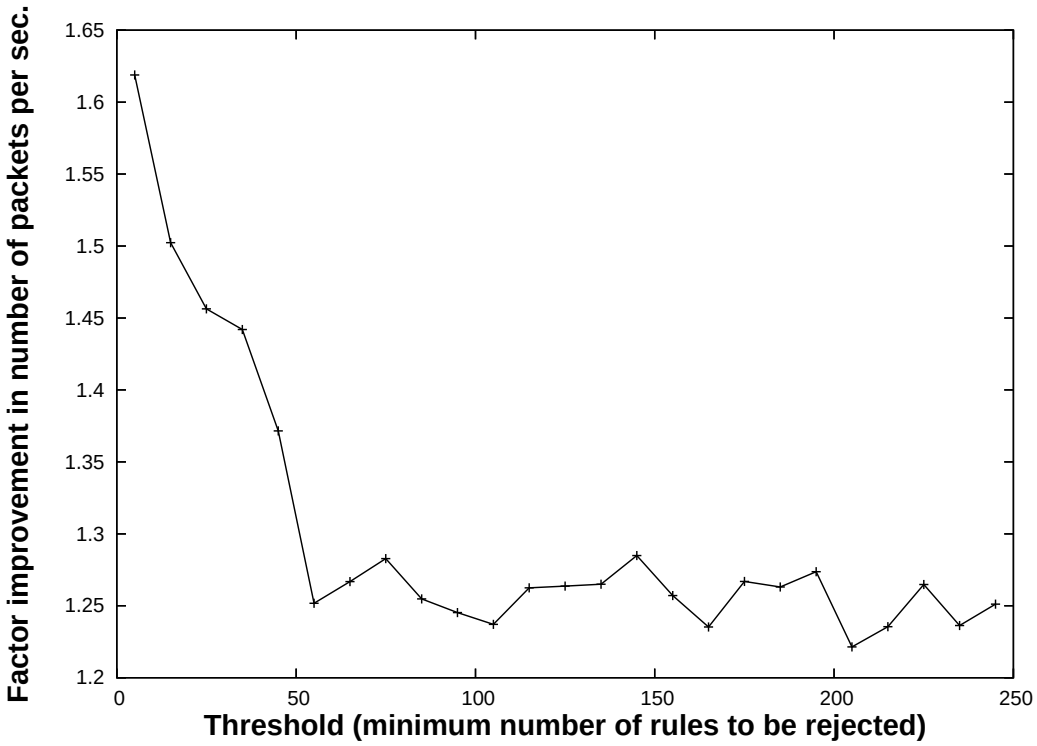


Figure 2.10: The change in number of packets processed with the threshold for minimum number of rules to be rejected, when compared to Snort (dataset: 98-test-w1-mon).

2.3.4 Variation with Threshold

In order to investigate how the threshold affects the performance of our system, we evaluated the DARPA dataset, 98-test-w1-mon, for different values of the threshold. Figure 2.10 shows the performance variation of our system with the increasing threshold. As expected, the performance of the system decreases with increasing cost assigned by threshold. However, we find that the changes are more pronounced only for lower threshold values. We find that the memory saved by our system increases with increasing threshold values, significantly only for lower threshold values. Therefore, we find that increasing the threshold reduces performance, but saves more memory, and this difference is more pronounced for lower threshold values.

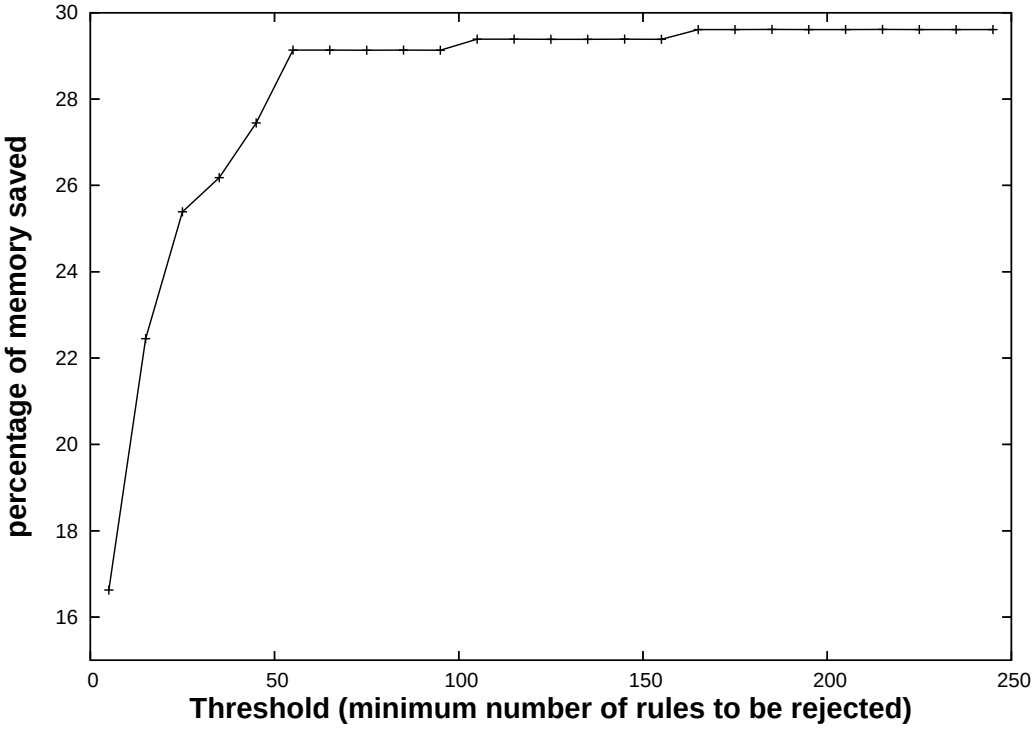


Figure 2.11: Variation in memory saving with the threshold for minimum number of rules to be rejected, when compared to Snort (dataset: 98-test-w1-mon).

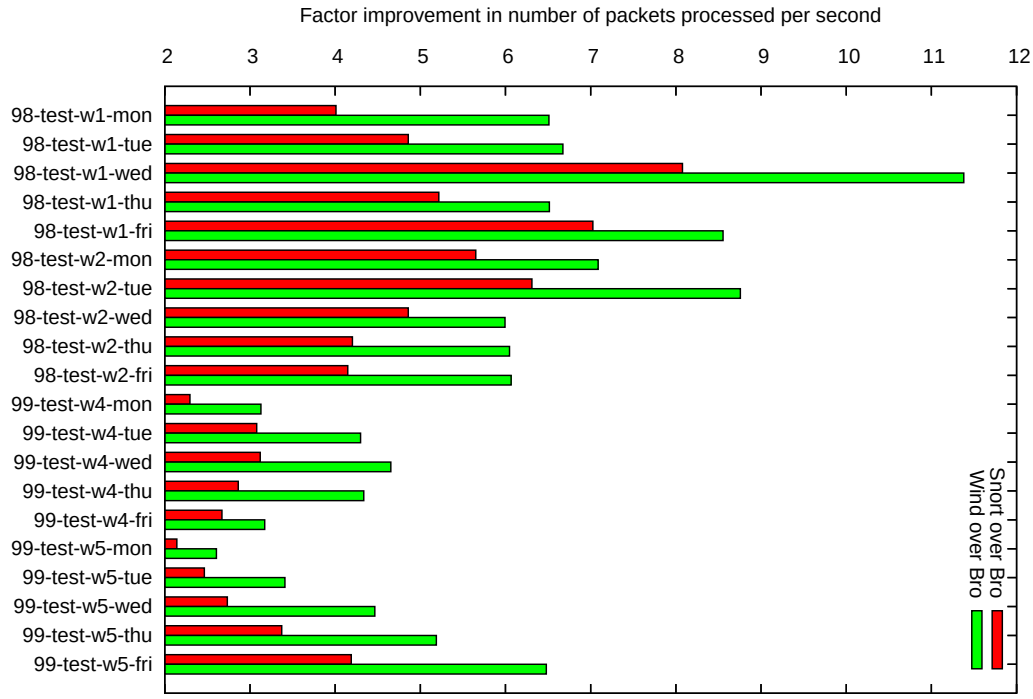


Figure 2.12: Factor improvement when comparing Bro with Snort and Wind for the number of packets processed per second (dataset: 99-test-w1-wed).

2.3.5 Comparison with Bro

We also compared Wind with another IDS Bro [60]. We first converted Snort signatures using a tool already provided by Bro [75]. However, only $1,935$ signatures were converted and regular expressions in the rules were ignored. We then compared Bro 0.9 with Wind and Snort for various DARPA workloads. As shown in Fig. 2.12, Snort is faster than Bro by 2 to 8 times, and Wind is 3 to 11 times faster than Bro. This result is partly because Bro uses regular expression for signature specification rather than Snort, which uses exact substrings for signature matching. Bro uses a finite automata to match regular expressions [75], whereas Snort uses the Wu-Manber algorithm for matching sets of exact substrings.

2.4 Dynamically Adapting to Changing Workload

The Wind system that we have described so far analyzes observed network traffic and input rules to speed up the checking of network packets in an IDS in a memory-efficient way. However, traffic characteristics can change over time, or the rule set can change as new vulnerabilities are announced. Therefore, we need to adapt our evaluation structure dynamically without restarting the system.

To adapt to changing traffic characteristics, we plan to collect traffic statistics in the intrusion detection system itself, and reorganize the evaluation structure when necessary. It would be too intrusive and costly to update statistics for each packet. Therefore, one could update statistics for a small sample of incoming packets. Then, we can use these statistics to determine the utility of specific groups in the structure, and determine the benefit that rules in the generic group would provide if they are separated from other rules in the generic group. We can then remove specific groups whose utility decreases over time and make new groups for rules in the generic group that provide increased benefit. However, to ensure the correct application of rules, these changes may require updating a portion of the evaluation tree atomically, thereby disrupting the incoming traffic. Therefore, one could develop algorithms that use the updated statistics to dynamically detect a significant change in traffic and trigger reconfiguration of the structure when the benefits far outweigh the disruption.

Vulnerabilities are announced on a daily basis. Sometime a number of vulnerabilities for a single application are announced in a batch, demanding a set of rules to be updated with the intrusion detection and prevention system. One naive solution is to add the set of rules to the existing evaluation structure, and then let the reconfiguration module decide over time if there is a need to create additional groups. However, this strategy may affect the performance significantly if a large set of rule is added to the generic group. This performance degradation would continue till new groups are created. Therefore, one could add rules whose values match with already existing groups directly to those specific groups. If a large number of rules still remain to be added to the generic group, then we can use our algorithm described in this chapter to determine the groups that should be separated. Then, additional groups can be created within the existing structure and the new rules added into those groups.

2.5 Conclusions and Directions for Future Work

In this chapter, we have argued that an intrusion detection and prevention system should adapt to the observed network traffic and the input rules, to provide optimized performance. We have developed an adaptive algorithm that captures rules and traffic characteristics to produce a memory-efficient evaluation structure that matches the workload. We have implemented two distinct components over Snort to construct a workload-aware intrusion detection system. The first component systematically profiles the input rules and the observed traffic to generate a memory-efficient packet evaluation structure. The second component takes this structure, pre-processes the rules, and matches any incoming packet. Finally, we have conducted an extensive evaluation of our system on a collection of publicly-available datasets and on live traffic from a border router at a large academic network. We found that workload-aware intrusion detection outperforms Snort by up to 1.6 times for all Snort rules and up to 2.7 times for web-based rules, and consumes 10-20% of less memory. A Snort implementation of Wind outperforms existing intrusion detection system Bro by six times on most of the workloads.

In this chapter, we demonstrated that deployment context can be used to improve performance of signature-based systems that detect *known threats*. In the next chapter, we show how accuracy of a *new threat* detector like honeynet is impacted by surrounding network and then present an automated approach to honeynet configuration that improves the accuracy of honeynets.

CHAPTER 3

Network Aware Honeynet Configuration

This chapter explores the importance of deployment context for capturing new threats to a network. In particular it examines why it is important for a new threat detector to be aware of its deployment and how can it automatically take deployment context to provide more accurate threat detection.

A number of tools [26, 53, 54, 55] have been developed to instrument live hosts and to detect new threats. The fundamental problem in instrumenting live (i.e., production) hosts is that such instrumentation affects the performance and availability of these systems. A security approach that alleviates these problems is to create a non-productive host (also called a honeypot) and place it at an unused or dark address. Honeypots are host and network resources that do not have production value and so any interaction with a honeypot is suspicious. For example, a honeypot deployment might consist of a mail server that is configured identical to a production mail server and placed nearby to detect possible intrusions against the real mail server. Because no legitimate clients are configured to use the honeypot mail server, any traffic to it is the result of malicious activity or misconfiguration.

Individual honeypots provide excellent visibility into threats that affect specific host and operating system configurations. However, detecting threats quickly with only a few honeypots is difficult. In order to quickly capture threat details, a number of commercial and non-profit organizations [4, 79], academic projects [16, 51, 83, 88], and tools [14, 62] have started monitoring large numbers of unused and dark addresses. We collectively call honeypot systems that monitor multiple unused and dark addresses *honeynets*.

Unfortunately, as systems have moved from a single honeypot to a network of honeypots, the problem of determining the configuration of these systems has largely been ignored. A single honeypot is easily configured with the operating system and applications matching an existing production system. However, for a large number of addresses, the network administrator must make decisions about what systems they want to protect and what attacks they wish to observe. For example, a single academic network in this study with 5,512 hosts had over 1,386 unique host and network configurations (see Table 3.4), and over 3000 unused addresses available for monitoring [23]. This effort is further complicated as the number of vulnerable services and potential attacks increase on the Internet. For example, Symantec documented 1,896 new software vulnerabilities from July 1, 2005 to December 31, 2005, over 40% more than in 2004 [78]. In spite of the difficulty of the task, little work has been done on how to automate honeynet configuration, leaving administrators to perform the task manually, and often in a generic or ad hoc fashion.

In this chapter, we show the potential limitations of generic and ad hoc approaches to honeynet configurations, and demonstrate that they lack visibility into network attacks. As an example, Table 3.1 shows that the most popular service on an example network is SSH, but attackers are focused primarily on nonexistent services such as Microsoft SQL Server. The honeynets in this network report neither of the two as the most critical security threat, indicating NetBIOS instead. As the example illustrates, these system do not represent the services running on the network (i.e., the vulnerable population) and fail to capture the existing attacks observed at the network (i.e., the threat landscape). Furthermore, we show that these honeynet configurations present very unusual operating systems and service configurations in many production networks and are trivially easy to detect, avoid, and corrupt (i.e., fingerprint). However, configuring honeynets is not trivial, as we find that there is lack of generality in vulnerable populations and threat landscapes over time and across networks. Furthermore, the potential number of addresses to configure may be very large and therefore, we require automated processes to configure a honeynet.

In order to automatically generate honeynet configurations with the goals to provide visibility into the vulnerable population and avoid detection, we argue that honeynets should be configured with individually consistent hosts and proportionally represent the surrounding network. We provide a simple, automated approach based on profiling the network and random sampling to generate honeynets with individual host consistency and proportional representation of the network. We

evaluate our approach by applying it to live networks and deploying the configurations generated. We find that the honeynets achieve the desired goals of providing an accurate view of threats to the networks and resistance to discovery. For example, the honeynets representative to academic networks accurately show that SSH brute force attacks are the most widely impacting threat, and those representative to web server farms accurately show web proxy attacks as the most widely impacting threat during the period analyzed.

To summarize, the main contributions of this chapter are:

- We show that ad hoc configurations do not provide visibility into the vulnerable population or into the threat landscape, and they are highly unusual in many networks, causing easy detection and fingerprinting.
- We justify the need for automatic configuration methods and then identify visibility and resistance to fingerprinting as the two goals of such a configuration system.
- We develop a simple technique that achieves the two critical goals automatically and we evaluate this technique through network monitoring and deployment of honeynets in production networks.

The remainder of this chapter is structured as follows: Section 3.1 presents the existing approaches to honeynet configuration and Section 3.2 shows the limitations of existing approaches. Section 3.3 provides the compelling reasons for automatically generating configurations, and discusses individual host consistency and proportional representation of the network as important properties that make such a configuration. Section 3.4 describes our algorithm for honeynet configuration and its evaluation. Section 3.5 concludes our work with directions for future work.

3.1 Background

The way in which honeynets are configured greatly impacts their effectiveness. In an effort to understand how honeynets are currently configured, we have examined publicly available honeynet deployment data. We find that these deployments are often tied with a specific honeynet tool or technique. Each of these tools, in turn, has its own set of configuration capabilities. One of the main factors that distinguishes each of these honeypot systems is their level of interactivity [76].

Interactivity defines the degree to which a honeypot behaves like a real end-host system, as a broad spectrum of end-host behaviors can be emulated in order to capture increasing amounts of attack details. Therefore, we group these deployments by their level of interactivity and discuss the configuration capabilities of each as well as their limitations.

On the lowest end of interactivity, a passive monitor logs incoming traffic to unused addresses without any response [51]. For example, the Team Cymru [27] and Moore *et. al.* [51] have deployed passive monitors in unused addresses. Passive monitors do not allow for any response and have no service configurations. Because they do not emulate any services, they provide limited visibility into threats and require advanced techniques to fingerprint [64].

A light-weight TCP responder replies with a TCP SYN-ACK packet for every TCP SYN packet received on every TCP port in order to recover the first payload from TCP worms [16]. The Internet Motion Sensor project [25] has deployed a lightweight TCP responder in 28 monitored blocks within 18 networks, with sizes of these blocks varying from /24 to /8 [15]. These systems emulate part of the network stack, accept incoming connections on all ports, but provide no application-level response. They provide minimal visibility into all applications, but are trivial to find. We call this configuration **All TCP Responder**.

Honeyd emulates network stacks of multiple hosts on a single machine and has a plug-in system for service emulation modules [62]. The Brazilian Honeynet Team [77] deployed numerous sensors of sizes from /28 to /24 with Honeyd, and the German Honeynet Team has also deployed a /18 honeyd sensor [37]. Finally, the French Honeynet Project [34] and the Norwegian Honeynet Team [57] have also deployed Honeyd sensors. Honeyd has a great deal of flexibility in honeypot configuration, enabling arbitrary personalities to be specified for each host. We call the default Honeyd 1.1 configuration [76] **Generic Honeyd**. To simplify configuration of large addresses, Honeyd allows declaration of default host configuration and ties it to all dark addresses that have not been assigned any host configuration. To automatically generate Honeyd configurations, RandomNet [74] allows operators to choose operating systems and emulated services, and constructs hosts with random combinations of chosen OSs and services. We call this configuration **Random Honeyd**.

Nepenthes emulates certain vulnerabilities to recover more exploits for the vulnerabilities [14]. The Chinese Honeynet Team has deployed a /24 Nepenthes sensor [20], the UK Honeynet Team [80] has deployed two Nepenthes sensors, the German Honeynet Team has deployed a /18 Nepenthes

sensor [37], and the Georgia Honeynet Team [36] has started experimenting with Nepenthes. While some configuration of these systems exists, as various vulnerability modules can be turned on or off, most deployments are limited to those modules that already exist. We call the default Nepenthes 0.1.7 configuration **Nepenthes**.

Numerous end-host systems can be run on the same hardware platform by using virtual machine tools such as VMWare [61]. A variety of academic projects have deployed VMWare-based solutions including Georgia Tech [28], UCSD [83], and Purdue [41]. The configuration of end-hosts is limited only by the number of operating system and application compatibilities. However, a common approach is a vanilla operating system install, such as those promoted in [61].

In summary, we find that most published honeynet deployments use well-known techniques for honeynet construction. These techniques have a variety of configuration characteristics that range from no configuration to a fixed configuration and finally, to fully configurable. When configuration options exist, there is no automated system to choose between these options and are left to the administrator for manual configuration.

3.2 Limitations of Ad Hoc Honeynet Configurations

One of the original goals of deploying honeynets is to provide protection to an organization by understanding the means and motives of attackers. However, this important goal can be achieved only if the honeynets are configured to represent the network and services. In the previous section, we showed that the configuration of honeynets has largely been ignored by existing deployments and tools. In this section, we argue that ignoring this problem is dangerous because lack of proper configuration impacts both the visibility that these honeynets can provide as well as their ability to avoid detection.

3.2.1 Impact of Ad Hoc Configuration on Visibility

In the previous section, we observed that, while some flexibility exists in how honeynets can be configured, the absence of automated techniques has caused this process to be manual. If careful attention is not paid to this manual process, these configurations can drastically impact the effectiveness of the monitoring systems. The effectiveness depends on the honeynet ability to protect

TCP Port	% of Vulnerable Population	% of Attacks	Honeynet Configurations		
			% of **Nepenthes** hosts	% of **Generic Honeyd** hosts	% of **Random Honeyd** hosts
21	24	-	100	4	45
22	53	-	100	3	-
23	34	-	-	3	65
80	26	0.4	100	4	62
106	14	-	-	-	-
135	9	0.6	100	-	-
139	17	0.5	100	96	74
311	14	-	-	-	-
427	26	-	-	-	-
445	10	1.4	100	1	40
497	28	-	-	-	-
554	-	0.9	-	-	-
1080	-	1.7	-	-	-
1433	-	6.8	-	-	-
1521	-	0.9	-	-	-
4444	-	0.5	-	-	-
5900	24	0.9	-	-	-

Table 3.1: The lack of visibility into both the threat landscape, as well as the vulnerable populations for a variety of different honeynet configurations.

important network resources from the most likely threats. As a result, improperly configured honeynets can fail to be effective if they can not accurately represent:

- **the vulnerable population.** The vulnerable population is the set of services on the network that can be remotely accessed. It is important to note that this definition of vulnerable population includes all services on the network rather than only those with known vulnerabilities.
- **the threat landscape.** The threat landscape describes the current attacks on the network.

To understand the impact of ad hoc configurations on visibility, we compared them to the threat landscape and the vulnerable population. Table 3.1 shows the top 10 open TCP ports found on an academic /16 network together with the top 10 attacked ports on a /24 honeynet in the network, and several honeynet configurations. TCP ports represent broad characterization of the vulnerable population, the threat landscape, and the threats a honeynet configuration can capture. **All TCP Responder** provides minimal visibility into all ports and is not shown in the table. The results

of the comparison are rather startling. *The ad hoc configurations of honeynets do a poor job of representing both the attacks to the network as well as the potentially vulnerable population.* This lack of visibility has profound implications, as the monitoring system will fail to provide any insight into attacks at large numbers of vulnerable hosts (e.g., those running TCP/106, TCP/311, TCP/427). In addition, these systems fail to capture the attacks prevalent on the network (e.g., those targeting TCP/1433, TCP/1080, TCP/1521, TCP/5900) and may make certain attacks (e.g., TCP/139) appear to be more important on the network than they in fact are. Interestingly, it also appears that attackers do not necessarily target the most popular services. In fact, only a small number of common ports are actually popular across the threat landscape, vulnerable population, and configurations including 80, 135, 139, and 445. We will show in the next section that the threat landscape and the vulnerable populations change over time and networks, and these honeynet configurations will have difficulty in representing them too.

3.2.2 Impact of ad hoc configuration on fingerprinting

As we saw in the previous section, ad hoc honeynet configurations do not represent either the exploit space or the vulnerable population. In addition to reducing important visibility, this behavior can be effectively used by attackers to detect, or fingerprint, the locations of the honeynets within a network. This type of activity is important to defenders, as fingerprinted honeynets can be actively avoided or corrupted by attackers. The key insight that makes this fingerprinting possible is the observation that some combination of services and operating systems on an ad hoc configured honeynet may seem highly anomalous when compared with the rest of the network. In the next two sections, we discuss a novel technique for fingerprinting honeynets and show how ad hoc configurations are easy to spot within productions network using this technique.

3.2.2.1 Fingerprinting Algorithm

Honeynets configured in an ad hoc fashion provide inconsistent view of services on a network. Because we do not know of any systematic approach to detect such configurations in a network, we developed a general approach that involves comparing service configurations within a subnet to the configuration of the entire network. A naive approach, such as examining each individual host in the subnet to see if it is anomalous with respect to the rest of the network, will only determine if that host

has a unique configuration in the network. In order to compare collections of host configurations, we first perform active tests to detect host configurations, aggregate these configurations by common operating systems and services, and then compare the distribution of a new service in the subnets with the network as a whole.

Consider a set of tests $\{T_1, T_2, \ldots, T_n\}$ that can be performed on an Internet host, with the result of test T_i being any member of the set $R_i = \{r_i^1, r_i^2, \ldots, r_i^{m_i}\}$ of size m_i. For example, a test (or variable) that checks whether TCP port 139 is open or not on an Internet host has results (or values) in the set $\{open, closed\}$ of size 2. The combination of test values for an Internet host makes a *host profile*. For example, checking port 139 and port 445 on an Internet host might have host profiles in the set {(closed, closed), (closed, open), (open, closed), (open, open)}. While the algorithm is test-set independent, in this chapter we utilize three sets of tests. Tests constructed based on existing tools, such as nmap [35], easily differentiate TCP software by exploiting ambiguities in RFCs and implementations, and we call them **tcp**. We similarly developed a set of 26 new tests to identify a web server and its configurations and call them **http**. Finally, we test open TCP ports from 1-1024 with a simple program that attempts connection to these ports that we call **ports**.

Our main idea is to group hosts by test values and then compare the subnet with the network as a whole for the values of a new test. First, we order tests by the increasing value of their entropy, each of which is computed by the distribution of test values among the hosts in the network. The tests are ordered by increasing entropy because a test with lower entropy has most certain values for the hosts on the network and hence, a significant anomaly can be detected if a subnet differs from the dominant values found in the network. Second, we compare each subnet with the network for the values of first test, then we aggregate by values of this test and compare for values of the second test. We iterate this comparison between the subnet and the network over the determined test order until the last test.

To quantify anomalies for a new test, the values for the test is arranged on a normal distribution curve using the frequency associated with each value. Then, we measure the anomaly using a *test of significance* called ***z*-statistics** [33] because of three specific reasons. (1) The anomaly quantified by z-statistics is significant if the values of a new test for hosts on the subnet are *significantly* different from those on the network. (2) If the network exhibits significant variation in the values of the new test chosen for analysis, then the anomaly is of low magnitude. (3) If only a few hosts in the

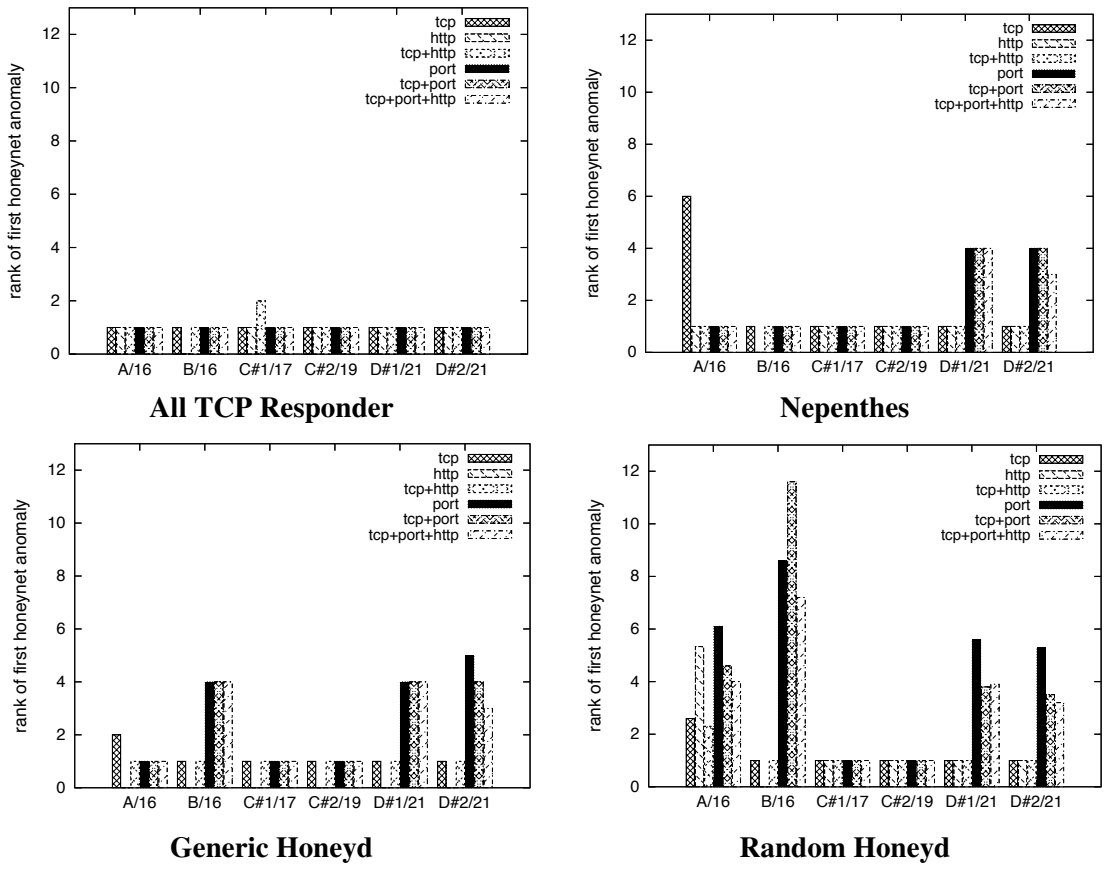


Figure 3.1: The rank of the first anomaly with ad hoc honeynet configurations in six networks for various combinations of tests that identify operating systems and services. Most honeynets represent the first, or most, anomalous subnet in a network

network are aggregated by test values, then it causes an anomaly of smaller magnitude for the new test values. For a formal definition and a small discussion of z-statistics, please refer to Appendix A. A complete illustrative example of the above algorithm is presented in Appendix B.

There are a wide variety of techniques for fingerprinting honeypots. Our goal is not to enumerate all possible attacks but rather to illustrate the importance of configuration artifacts in honeynet fingerprinting. One relevant approach for detecting honeypots involves probing and analyzing published records from honeypots [18] . However, this attack is possible only when attack statistics from honeypots are publicly available. Rajab *et. al.* [64] fingerprint passive darknets that do not respond to any probe whereas we discover active honeynets without a prior knowledge of how they are configured. This is achieved by comparing subnet configurations with the surrounding network and detecting anomalous configurations.

3.2.2.2 Detecting Ad Hoc Configurations

In order to study the impact of ad hoc configurations in terms of their ability to be discovered, we embedded different honeynet configurations into six network (A/16 and B/16 are academic networks, and C#1/17, C#2/19, D#1/19 and D#2/19 are web-server farms) profiles. The network profiles were created by probing live networks and storing their configurations. The honeynet configurations were created by a variety of existing approaches. We then applied our fingerprinting algorithm to create a list of anomalous subnet configurations in each of these networks and sorted them into descending order of how anomalous they were (i.e., by z-values). Ideally, each of the embedded subnets (i.e., honeynets) should appear on the top of these lists. Therefore, we will evaluate a honeynet resistance to fingerprinting by the position (or rank) in the lists.

Figure 3.1 presents the rank of the first honeynet anomaly in six networks when the honeynet is configured using **All TCP responder**, **Nepenthes**, **Generic Honeyd** and **Random Honeyd** (average of ten random honeyd configurations). We find that the ad hoc honeynet configurations surface as the most anomalous configurations in almost all of the networks for many set of tests. This result is perhaps easy to realize for honeynets such as **All TCP Responder** with all ports turned on, but quite surprising for honeynets configured with individually consistent hosts (i.e., they look like a real host). This is because **Nepenthes**, **Generic Honeyd**, and **Random Honeyd** often emulate services or run service combinations that are anomalous in many networks. As was the case with visibility, the impact of ad hoc configurations can be profound, allowing attackers to easily fingerprint honeynets.

3.3 The Properties of Honeynet Configuration

In previous sections, we have shown that existing methods for honeypot configuration are largely manual in nature. We have shown that if proper care is not taken in the construction of these configurations, they can produce honeynet systems that fail to provide visibility and that are easy to fingerprint. In this section, we describe the process of generating honeynet configurations. First, we motivate the need to automate honeynet configuration by showing the lack of temporal generality of exploits and spatial generality of vulnerable populations, the diversity of systems in networks, and the large number of addresses to be configured. Second, we present visibility into vulnerable

04/19/2006	05/19/2006	06/19/2006	07/19/2006	08/19/2006
6000	445	22	135	1433
445	139	5900	80	1080
1433	5000	3128	4444	445
1080	1433	8080	445	5900
135	80	80	1433	1521

Table 3.2: The top 5 TCP ports observed in a /24 sensor in network B/16, over a period of 5 months. Exploits change quickly over time.

population and resistance to fingerprinting as the two desired objectives and argue that a honeynet configuration with individually consistent hosts and proportional representation of the network will achieve the desired objectives.

3.3.1 Need for Automatic Configuration

While we have clearly shown that ad hoc configurations are a danger to the goals of any honeypot deployment, we have not yet shown that the process of generating these requires anything more than careful manual configuration. We believe that two properties of the configuration space motivate the need for automatic configuration:

3.3.1.1 Lack of Generality

A major impact on the effort of manual configuration of honeynets is the lack of temporal generality of attacks. Table 3.2 shows the top five TCP ports and the number of packets observed over a day for last five months on the /24 honeynet in the B/16 network. We find that new services were targeted heavily each month. The TCP ports 6000 and 1080 were the unusual ones targeted in April, the TCP port 5000 was targeted in May, the TCP ports 22 and 5900 were targeted in June, and TCP port 4444 in July. Moreover, the exploits observed vary significantly with locations, as demonstrated by significantly different data observed on sensors deployed in different networks [24]. Therefore, a highly variable nature of threat landscape makes chasing the exploits really difficult for the defenders, who must manually configure their honeynets.

Spatial generality also impacts the effectiveness of manually generating honeynet configurations. Differences in attack behaviors and vulnerable populations across networks make sharing of configuration a near impossibility. For example, Table 3.3 compares vulnerable population in

Operating System	Networks			
	A/16	B/16	C#1/17	D#1/19
Windows	44	25	76	77
Cisco IOS	14	7	-	-
Apple	9	36	-	-
Linux	9	7	15	6
HP printer	3	13	-	-
Solaris	9	7	1	2
*BSD	1	-	8	15

Operating Systems

TCP Port	Networks			
	A/16	B/16	C#1/17	D#1/19
139	42	17	-	-
22	41	53	30	25
135	39	10	42	69
23	27	34	4	5
445	27	11	-	-
80	21	26	93	96
25	12	10	70	83
21	8	24	77	79
427	4	26	-	-
497	3	28	-	-
110	1	-	39	17

TCP ports

Table 3.3: Comparing the vulnerable population in four networks, by operating systems and TCP ports. The vulnerable populations are different across networks.

various networks in two ways, by the operating system and the TCP ports. The second largest operating system in network A/16 is surprisingly Cisco IOS, which is found in the wireless access points and routers in the academic campus. On the other hand, Apple Mac OS is the dominant operating system in network B/16. As expected, the web-server farms were dominated by Windows servers. While SSH seems to be predominant service found in A/16 and B/16, HTTP, FTP and SMTP seems to be the dominant services in the web server farms. Therefore, the vulnerable population varies significantly with the nature of the networks making it difficult for a single honeynet configuration to represent the vulnerable population across networks.

3.3.1.2 Scale of Configuration

Traditionally, a small number of individual honeypots were configured manually. For example, a dedicated host will be configured as a clone of a valuable network resource, such as a mail server. However, as the number of services and the number of hosts running them on the network have increased, defenders have been under increasing pressure to manage and provide visibility into these new services. Unfortunately, manual honeynet approaches for achieving this visibility are difficult to apply. Consider the data presented in Table 3.4, which shows surprisingly large numbers of TCP implementations, HTTP implementations, services, and the complex associations between them in

Network	Type of organization	Hosts	Web-servers	TCP	HTTP	TCP+ Ports	TCP+ HTTP	TCP+ Ports+ HTTP
A/16	university network	5512	1237	352	241	1210	699	1386
B/16	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 3.4: The number of unique host configurations observed at six production networks for various tests and their combinations. Each network has a surprisingly large number of unique configurations.

various production networks. For example, in network A/16, we find that the 5,512 hosts result in 352 unique TCP stack implementations. Of those 5,512 hosts, 1,237 were running Web servers, and when probed, yielded 241 differing servers and configurations. This included TCP stacks and Web-servers from a bewildering variety of devices including NATs, wireless access points, printers, power switches, and webcameras.

In addition to the diversity in the numbers and types of individual host configurations, network administrators are also required to configure a potentially large number of unused addresses in order to maximize visibility. Traditional honeynet deployments of statically allocated network blocks are giving way to dynamic discovery of all unused addresses [23]. The number of addresses available can be surprisingly large, even in small organizations.

3.3.2 Individual Host Consistency and Proportional Representation

A number of objectives can be used to define honeynet configurations. For example, one may choose the easiest set of configurations to deploy, or one may want to represent global host or threat distribution. In this chapter, we set our objective as providing visibility into the vulnerable population on the defender's network and resistance to fingerprinting. We chose these objectives so that the honeynet provides intelligence into zero-day threats to the network quickly. Even though we chose specific objectives, we believe that the general approach described in this chapter (sample and configure) can be applied to configure honeynets representing other distributions (e.g., threat).

To provide visibility into network threats and resistance to fingerprinting, we argue that a honeynet configuration must have individually consistent hosts and should proportionally represent the

network. An individually consistent host is one whose configuration appears in a live network. Individual consistency is necessary for two reasons. First, in order to provide visibility into threats that impact the defenders network, the configuration must provide the same opportunities for attackers as they occur in the real network. It is obvious that in order to capture threats against a service, you must run that service. Perhaps less obvious is that sophisticated worms, like Slapper, attempt to identify specific services and versions of operating systems to deliver the correct exploit [66]. In addition to visibility, individual host consistency provides resistance to fingerprinting against attackers that look for unusual services on the host.

In addition to individual consistency, we argue that visibility and resistance require proportional representation of the network. Honeynets are primarily deployed for detecting new threats. However, we do not know the operating systems and the service configurations that will be targeted by a new attack. Since we cannot anticipate any particular configuration, the ratio of honeynet hosts for any combination of operating system and services should be equal to the ratio of network hosts with those combinations. Proportionality provides visibility in that the most prominent software on the live network appear more frequently and hence can capture new threats to the software quickly. This also helps in prioritizing attacks by their widespread possible impact on the network, and hence arranging attacks by the maximum number of honeynet hosts that observed each one of them. Furthermore, when configuring large addresses with realistic operating systems and services, proportionality ensures resistance to fingerprinting in that the system will not add a number of configurations that skew the distributions of these configurations in the network as a whole.

3.4 A Simple Technique for Honeynet Configuration

Our algorithm for generating representative and consistent configurations has two components: profiling the network and generating configurations. Profiling the network is the process of determining the configuration of the existing production network for which we want visibility. After profiling the live network, we need to determine host profiles to be deployed on the honeynet. While we limit our discussion to the widest vulnerable population, our approach is also valid for specific vulnerabilities scanned by a tool such as Nessus [5].

The generated host profiles should be individually consistent and proportionally represent the network for all combination of tests values (as they identify operating systems and services). For

example, consider a network with three hosts whose profiles for port (139, 445) tests are [(open, open), (open, closed), (closed, closed)]. A honeynet proportional to this network should meet multiple constraints. When each port is considered separately, $2/3$ of their hosts should have port 139 open, and $1/3$ of their hosts should have port 445 open. When the ports are considered in combination, $1/3$ of their hosts should have both ports open, $1/3$ of their hosts with port 139 open and port 445 closed, and finally $1/3$ of their hosts with both ports closed [1]. Due to resource (address space and machines) constraints, a honeynet may only approximate the network. Therefore, we need to develop an algorithm that best approximates proportional representation for all combinations of test values, and at the same time, produces individually consistent profiles. However, it is difficult to construct an algorithm that minimizes error in proportional representation for *all* possible combinations of test values. Surprisingly a simple technique to select a *random sample* [22] of hosts in a network ensures our two critical properties:

- **Proportional representation.** The probability that a combination of test values $(T_{i_1} = r_{i_1}^{j_1}, T_{i_2} = r_{i_2}^{j_2}, \ldots, T_{i_k} = r_{i_k}^{j_k})$ is selected by the sampling algorithm is equal to the fraction of live hosts that satisfy them. Therefore, simple random sampling chooses host profiles with a probability distribution present in the actual network and approximates it in the selection of hosts. If n hosts are picked from a total population of N, then the standard error in estimating the ratio on the honeynet for a combination of test values with standard deviation σ is equal to $\sigma\sqrt{(\frac{1}{n} - \frac{1}{N})}$ [22]. Therefore, as more and more hosts are picked up to be represented on the honeynet, the better the honeynet represents the actual network.

- **Individual host consistency.** To be effective in capturing threats and in warding off possible fingerprinting efforts, we would like the configuration for each host on the honeynet to be individually consistent. This means that none of the hosts should be assigned an unusual configuration that is not possible in the real world. Since a configuration is assigned to a host in the honeynet only if there is a host with that particular configuration in the network, every host in the honeynet has a realistic configuration. Hence, for a subset of configuration values, a honeypot host will match a number of live hosts, yet represent a very specific live host when all configuration values are considered together.

[1] an open port by may be preferred over a closed port, and we will discuss this later.

		% of Network Hosts	% of **Representative** Hosts
Operating Systems	Apple	36	35
	Windows	25	23
	HP printer	13	15
	Linux	7	5
	Solaris	7	6
TCP Ports	22	53	54
	23	34	34
	497	28	31
	427	26	27
	80	26	24

Table 3.5: Evaluating representative honeynet configuration by visibility into the vulnerable population. The percentage of vulnerable hosts for the top five services and top five operating systems in network B/16 match closely to the representative honeynet.

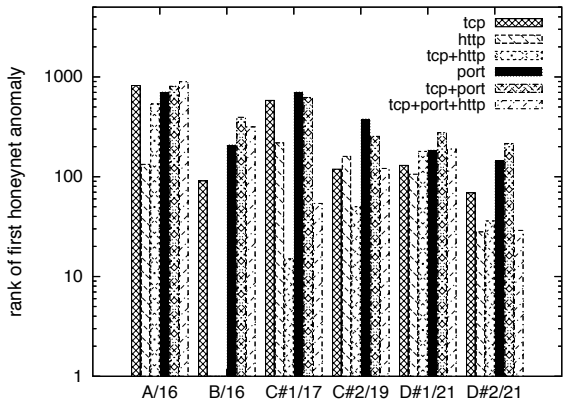


Figure 3.2: Evaluating representative honeynet configuration by resistance to fingerprinting. The representative honeynet is very resistant to fingerprinting.

The simple random sampling is effective because we do not know the host configuration that will be targeted by a new attack. However, when we know certain tests are more important than other ones, then we can achieve better proportionality to the important tests. For example, if we know that services are frequently being targeted, then we can achieve better proportionality for the service distribution than say the operating system distribution. This is usually achieved using *stratified sampling* [22]. Stratified sampling involves hierarchically separating the population and allocating the size to be sampled proportionally at each point in the hierarchy.

Previously we have assumed that all hosts in a network are equally important. Situations can easily arise in which this will not hold true. An operator might need better visibility to specific machines. For example, a system administrator may wish to weigh his DNS server, a single point of failure, twice when compared to other hosts on the network. To provide additional visibility to a machine when compared to others, additional matching configurations are added to the configuration pool before sampling. In addition, the operator might indicate a particular value for a test variable as more important than other values. For example, configuring a host with port80 *open* is more important than when it is *closed*. To account for these cases, we allow users to specify such preferences as input to tune honeynet configuration. Then a host is replaced by a preferred host before sampling. For example, if port 80 open is preferred over closed, then a Windows XP host might be replaced by Windows XP with IIS web server found in the network. This ensures that honeynet hosts are individually consistent even with user input.

Now we evaluate our approach to configuring honeynets. Recall that our objective was to create configurations that provide visibility into network attacks and resistance against fingerprinting. To demonstrate this, we validated our configurations by comparing with the vulnerable population and by applying the fingerprinting algorithm. To evaluate visibility of honeynets into real network threats, we created and deployed various honeynet configurations in production networks. Then, we examined the attacks observed by each of the honeynets, and show that non-representative configurations provide skewed views of the network threats. Finally, we deploy representative configurations for various other networks in our academic network and show that they provide different insights, even when deployed in a single network.

Rank	Exploit	% Network Hosts	% **Representative** Hosts
1	RPC portmap rusers request UDP	90	100
2	NETBIOS SMB-DS C$ share unicode access	7	10
3	NETBIOS SMB-DS IPC$ share unicode access	7	10
4	BARE BYTE UNICODE ENCODING	4	24
5	WEB-MISC robots.txt access	4	24

Table 3.6: The visibility of configurations into the exploits on the network. The percentage of network hosts that observed each of the top five exploits in network B/16 is compared with the distribution in the representative configuration.

3.4.1 Evaluating Correctness of Representative Configurations

In order to achieve visibility, we have argued that a configuration needs to provide proportional representation of the target network. Table 3.5 shows the visibility that a representative honeynet provides into the vulnerable population. We find that the distribution of network services and operating systems on the representative honeynet closely match the vulnerable population. The combination of operating systems and services between a representative honeynet and the network also matched closely and we omit those results here.

To evaluate honeynet configurations by their resistance to fingerprinting, the representative configurations for six networks were embedded in the networks and the anomaly detection algorithm from Section 3.2 was used on all of the networks. Figure 3.2 shows the rank of the first honeynet anomaly for six networks when the honeynet is configured to be **Representative** to these networks. It shows that a representative honeynet is very resistant to techniques that attempt to detect network-wide anomalies.

3.4.2 Evaluating Visibility of Representative Configuration by Network Monitoring

We demonstrated that the broad characterization of vulnerable population on the network matches the representative honeynet. In this section, we explore the visibility of representative honeynet into specific exploits on the network. For this, we capture exploits on the network B/16 using an intrusion detection system called Snort [67] and then compare honeynet visibility to those threats.

We monitored the gateway router to B/16 via a span port connected to a FreeBSD machine running Snort 2.1.3 configured with default signature distribution and recent bleeding-snort [2] sig-

Configuration	Exploits	% of Honeynet hosts	% of Susceptible Network Hosts
Representative	1. SSH Brute-Force attack	41	53
	2. NETBIOS DCERPC ISystemActivator path overflow attempt	8	9
	3. SHELLCODE x86 NOOP	6	9
	4. NETBIOS SMB-DS C$ share unicode access	1	10
	5. NETBIOS SMB-DS IPC$ share unicode access	1	10
All TCP Responder	1. NETBIOS DCERPC ISystem Activator path overflow attempt	60	9
	2. WEB-IIS view source via translate header	17	26
	3. P2P GNUTella GET	5	0
	4. LSA exploit	4	10
	5. SHELLCODE x86 NOOP	4	9
Generic Honeyd	1. SSH Brute-Force attack	8	53
	2. SCAN Proxy Port 8080 attempt	2	100
	3. BACKDOOR typot trojan traffic	1	100
	4. Behavioral Unusual Port 135 traffic	1	9
	5. Squid Proxy attempt	1	100
Random Honeyd	1. WEB-IIS view source via translate header	21	26
	2. LSA exploit	6	10
	3. FTP anonymous login attempt	6	24
	4. MS04011 Lsasrv.dll RPC exploit (WinXP)	3	10
	5. MS04011 Lsasrv.dll RPC exploit (Win2k)	3	10

Table 3.7: The impact of different configuration approaches into threat visibility. Non-representative configurations are misleading about the real threats to the network.

natures. The experiment was conducted for five hours on 2^{nd} May, 2006. The detected exploits were ordered by spread of the attack i.e., by the percentage of network hosts that observed the attack. We then grouped hosts that received a particular exploit and identified host characteristics necessary to receive the exploit. Finally, we analyzed the honeynet profile to discover susceptible hosts on the **Representative** honeynet. Table 3.6 shows the top five attacks (server-based) discovered on the network and compares it with the percentage of susceptible hosts on the honeynet. Ninety percent of the network hosts observed attempts to list currently logged users. This was targeting UDP port 111 and would have been visible on all honeynet hosts. Seven percent of network hosts observed administrative access attempts to SMB service on TCP port 445 and would have been visible on 10% of the honeynet hosts. The rest of the exploits were slow moving attacks and were observed on a few hosts on the network. Therefore, existing exploits on the network would be detected on the

A/16	C#1/17	C#2/19	D#1/19	D#2/19
1. SSH Brute-Force attack	1. Web Proxy GET Request	1. Web Proxy GET Request	1. NETBIOS DCERPC ISystem-Activator path overflow attempt	1.NON-RFC HTTP DELIMITER
2. NETBIOS DCERPC ISystem Activator path overflow attempt	2. WEB-IIS view source via translate header	2. NETBIOS DCERPC ISystem Activator path overflow attempt	2. SSH Brute-Force attack	2. NETBIOS DCERPC ISystem-Activator path overflow attempt
3. MS04-007 Kill-Bill ASN1 exploit attempt	3. NETBIOS DCERPC ISystem-Activator path overflow attempt	3. HTTP Challenge/ Response Authentication	3. WebDAV search access	3. FTP anonymous login attempt

Table 3.8: The importance of context. The top three attacks captured on honeynets representatively configured for five different production networks, but placed within the same network.

honeynet depending on their impact on the network.

3.4.3 Evaluating Visibility of Honeynet Configurations by Real Deployments

In this section, we deploy honeynet configurations in B/16 and evaluate them by the accuracy of threat view they provide. First, we compare threat observed on different honeynet configurations with the vulnerable population on the network. Second, we deploy representative configurations for various networks in B/16 and observe how representative honeynets provide unique view of threats to a network.

We deployed a real /24 honeynet (monitoring 256 addresses) with different honeynet configurations in the B/16 network and exposed it to the present day attacks. Each honeynet configuration was deployed with Honeyd 1.0 [63], for a period of one day using the service scripts for FTP, SMTP, FINGER, HTTP and IIS-emulator, and the entire experiment lasted from 22^{nd} April, 2006 to 1^{st} May, 2006. We evaluated the results across two dimensions: the affect of various configuration approaches on the threats observed, and the impact of context in representative configurations. It is important to note that comparisons across configurations in this experiment are not valid due to temporal changes in threat. Instead, our goal is to evaluate a honeynet configuration by comparing it with the vulnerable population.

Table 3.7 compares a honeynet representative of network B/16 with three other honeynet configurations. It shows the top five exploits, ordered by the number of honeynet hosts that observed the exploit. For the **Representative** configuration, we found that the SSH Brute-Force attack was observed on a large number of honeynet hosts and it matched with the university network B/16, which

has a significant number of hosts configured with SSH service. An attack on NETBIOS DCERPC service on port 135 was observed on a small percentage of hosts, as only 9% of the network hosts were susceptible to this attack. On the other hand, **All TCP Responder** observed a NETBIOS DCERPC attack on a 60% honeynet hosts. The **Generic Honeyd** configuration observed the SSH brute-force attack on only 8% of the hosts, but 53% of the hosts were susceptible to this attack. The **Random Honeyd** configuration observed an IIS attack on 21% of the hosts, which was close to 26% of Web servers in the network. However, around 50% of **Random Honeyd** hosts were capable of receiving the exploit (much higher than what actually received) but did not receive it because of slow moving attacks. When examining these results, we find that the non-representative configurations bias the threat view of a network away from the exploits that are actually the most widely affecting, and hence most important.

To analyze how well a representative configuration reflected attacks on a network, we deployed honeynet configuration representatives for five other networks. Each of them were deployed for one day in the /24 unused address space in the B/16 network. Table 3.8 compares representative honeynet configurations with the susceptible network population for five networks. We find that the SSH Brute force attack was commonly observed on honeynet hosts, and it coincided with the large number of hosts in the university network A/16 that were configured with SSH service. The dominant exploits found on the web server farms were those impacting web service and the TCP port 135, which are the dominant services in those networks. What is interesting to note is that, although these configurations were placed within the same network, the configurations observed vastly different views depending on the network vulnerability spaces.

In this section, we have examined a variety of different aspects of visibility. We have shown that the representative honeynet provides accurate view of threats on the network than ad hoc configuration methods whose results are often misleading. We also demonstrated that representative configurations observes different threat views depending on the network vulnerability space.

3.5 Conclusion

In this chapter, we addressed the problem of honeynet configuration. We showed that the existing approaches to configure a honeynet are manual, causing honeynets to be configured either in a generic or ad hoc fashion. We demonstrated the limitations of generic and ad hoc configu-

ration in that they provide poor visibility into network attacks and they are easy to discover. We show that providing visibility into network attacks is not trivial because the threat landscape and the vulnerable populations change with time and across networks. Furthermore, large number of dark addresses available in an organization requires automated approaches to configure honeynets. We identify individual host consistency and proportional representation as two properties required to achieve the desired goals of providing visibility into vulnerable population and resistance to discovery. We then described an automatic approach based on profiling the network with active tests and random sampling that achieves the desired goals. We evaluated the efficacy of representative honeynets through deployment of these configurations in a production network.

In this chapter, we have explored and evaluated honeynet configurations only for public facing networks. However, the deployment of firewalls and NATs have led to the need for internal honeynets [23] and we plan to evaluate the effectiveness of our techniques in these types of deployment. There are numerous means of discovering honeynets, and we by no means attempt to address all of them. In this chapter, we only examined configuration artifacts that can be used to aid attackers.

Having demonstrated the utility of deployment context for honeynet configuration, we now show how network traffic characteristics impacts accuracy of reputation-based system like blacklist generation. We then show how traffic characteristics on a network can be automatically exploited for improving blacklist accuracy.

CHAPTER 4

Context-Aware Blacklist Generation

Compromised hosts on the network provide a resource-rich environment from which attackers launch denial of service attacks, host phishing sites, send spam, and perform a variety of other malicious activities. The scope of this problem is huge, with current estimates of the number of compromised hosts on the Internet ranging into the hundreds of millions [84]. The massive scale of these attacks and the diversity of attack methods have stymied existing security solutions. For example, a recent evaluation of popular anti-viruses showed that they detected less than 60% [58] of recent attacks.

Acknowledging this lack of effectiveness, defenders have begun looking for new mechanisms to deal with the increasing number of compromised hosts. One technique becoming increasingly popular, especially in the network operation community, is that of reputation-based blacklists. In these blacklists, URLs, hosts, or networks are identified as containing compromised hosts or malicious content. Real-time feeds of these identified hosts, networks, or URLs are provided to organizations who then use the information to block web access, emails, or all activity to and from the malicious hosts or networks. Currently a large number of organizations provide these services for spam detection (e.g., NJABL [6], SORBS [9], SpamHaus [11] and SpamCop [10]) and for intrusion detection (e.g., DShield [81]). While these techniques have gained prominence, little is known about their effectiveness or potential draw backs.

We examined the effectiveness of four popular reputation-based blacklists [73] that are used for spam detection. We found that these blacklists exhibit a significant number of false negatives and a non-trivial amount of false positives. SORBS had a false negative rate of 65% and SpamCop and

Spamhaus had false negative rates of around 35%. We also found that SORBS exhibited a false positive rate of 10% after it blacklisted six prominent Gmail servers. We speculated that many of these false positives and false negatives may have been caused by the way the blacklists are generated. However, since very little is known about the approaches taken by these services to generate their blacklists, little research exists on the reasons for these failures.

To help answer these questions, this chapter examines a variety of methods for generating blacklists and evaluates these approaches through our own production spamtrap deployment covering 11 live domains and with access to a separate production email network covering thousands of hosts. We first explore a simple threshold-based approach to blacklist generation in which an IP address that exceeds a threshold number of spams sent to a spamtrap is blacklisted. As a representative technique for generation, we show that the method fails for a variety of reasons including: a lack of local awareness that leads to blacklisting popular servers like Gmail, targeted spam that misses the spam traps, low volume spammers that slide under the thresholds, and the necessary delay between the spam hitting a spam trap and the publication of the blacklist.

To address the problems of the threshold-based approach, we propose a new context-aware method for blacklist generation. By making use of the local email usage, reachability information, and the global information provided by spamtraps, we can provide a significant improvement over existing approaches. In particular, this context-aware paradigm enables two specific techniques: ratio-based blacklisting and speculative aggregation. In the ratio-based blacklisting approach, the traffic on the live network is compared to the traffic on the spamtraps to determine if it is safe to blacklist an IP address. We call this approach the ratio-based approach as the ratio of email messages on the live network to the email messages on the spamtrap is used as a measure to blacklist an IP address. In the second approach, speculative aggregation, we use global information provided by spamtraps and local email usage to identify good and bad neighborhoods relative to a given network. Such identification enables better prediction of spam sources while limiting the chance that these predicted hosts or networks are of use to the local network.

To validate our technique of the threshold-based approach, the ratio-based approach and the speculative aggregation are evaluated on a production email system deployment, which received 2.5 million mails, and our own separate spamtrap deployment, which received 14 million mails, during the month-long evaluation period in February-March, 2009. We find that the ratio and speculative

context-aware approaches perform better than the threshold approach in terms of false positives and false negatives for all threshold values. We find that the detection rate for the ratio-based approach is three times that of the threshold-based approach for a false positive rate below 0.5% and the speculative aggregation approach provides five times the detection rate when compared to the simple threshold-based approach for a false positive rate below 0.5%.

To summarize, the main contributions of this chapter are:

- An investigation of spam prevalence and spam blacklist effectiveness on a large academic network. We show that current blacklists have significant amount of false negatives and non trivial amount of false positives.
- An investigation into the causes of inaccuracy of current blacklists and of the current blacklist generation techniques.
- We propose and evaluate a new paradigm for reputation-based blacklist generation called context-aware blacklists in which we address the limitations discovered. We argue that blacklist generation techniques should take into account both *local* usage and reachability information as well as *global* reputation data when making policy decisions. The new paradigm is shown to be significantly more effective than existing approaches.

The remaining chapter is structured as follows: Section 4.1 describes the background and work related to Internet blacklists. Section 4.2 presents an evaluation of four prominent blacklists. Section 4.3 presents an investigation into the causes of inaccuracy of current blacklists and of the current blacklist generation techniques. We present our context-aware blacklist generation technique in Section 4.4 and a detailed evaluation of the approaches in Section 4.5. Finally, we conclude in Section 4.6 and present future directions for reputation-based blacklist.

4.1 Background and Related Work

Access control devices, like firewalls, enforce reputation that is statically decided. In recent years, more powerful dynamic reputation-based systems in the form of blacklists have evolved. A number of organizations support and generate dynamic blacklists. These organizations include spam blacklist providers like SORBS [9], SpamHaus [11] and SpamCop [10] and the network-based

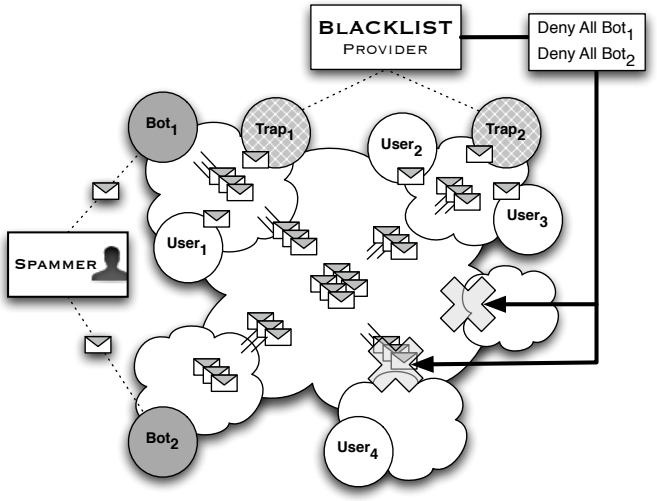


Figure 4.1: Existing approaches to blacklist generation. Blacklisting policy is set globally and enforced locally.

blacklist providers like DShield [81].

In order to understand how the blacklists are currently generated, we examined publicly available information on blacklist generation. Spam blacklist providers set up and monitor a number of unused email addresses called *spamtraps*. The spamtraps are deployed in two fashions. The first approach is to set up a mail server for an unused domain. For example, the Project Honeypot [82] takes unused sub-domains, like mail1.umich.edu, within legitimate domains, like umich.edu, and monitors all emails to these domains. The second approach is to use unused users within a legitimate domain. In this deployment model, the mail server delivers all emails directed to existing users to their respective folders but any email directed to a non-existant user is delivered to a separate account. We used the second approach to collect spamtrap emails and monitored non-existent users on 11 legitimate domains. Mails sent to spamtraps are then aggregated by a blacklist provider as shown in Figure 4.1. The mails are aggregated by source IP and then IP addresses that exceed a threshold number of hits in a time window are blacklisted. Since legitimate mail servers like yahoo.com can also be used by spammers, a threshold-based approach can put legitimate mail servers into a blacklist, causing wide-spread email disruption. So commercial blacklist providers may also maintain a white list and then use "Received" headers added by those legitimate servers to determine the IP addresses of the sender. This scheme does not work with Gmail because Gmail does not add the

source IP of the client if the web interface is used for sending the mail [73]. Further, the white list used by the providers is not publicly available. SpamCop also uses a sample of DNS lookups to determine if some IP addresses can avoid being blacklisted, which may not be a reliable estimate of actual mails delivered because of DNS caching. In our approach, we evaluate a context-aware approach in which the ratio of actual number of emails delivered in the network to the number of spamtrap hits is used for deciding blacklisting.

Recently a number of research papers have looked at the algorithms to generate blacklists. Ramachandran *et. al.* [65] proposed a new method to blacklist source IPs based on their mail sending patterns. However, their experiment is only based on mails received on the spamtraps and not on mails received on the live network. As a result, they only evaluate the false negatives of spamtrap received mail and not the false positives of their approach. In our study, we generate blacklists based on spamtrap mails and then apply them to the mail on the live network, so we evaluate the false positive and false negatives for the mail on the live network.

Xie *et al.* [87] have shown that a large number of IP addresses are dynamically assigned and mails from these IP addresses are mostly spam, so they recommend adding dynamic IP ranges into blacklists to reduce the false negatives.

DShield [81] aggregates intrusion detection alerts and firewall logs from a large number of organizations. It then publishes a common blacklist that consists of source IPs and network blocks that cross a certain threshold of events. Zhang *et al.* [89] argued that a common blacklist may contain entries that are never used in an organization. So they proposed an approach to reduce the size of the blacklists and possibly reduce the computational overhead in blacklist evaluation. However, they do not evaluate the effectiveness in terms of the false positive rate and the false negative rate of the blacklists.

4.2 Effectiveness of Current Blacklists

4.2.1 Experimental Setup

We monitored traffic using a traffic tap (i.e., span port) to the gateway router which provides visibility into all the traffic exchanged between the network and the Internet. The TCP streams on port 25 were reassembled using libnids [85]. The data sent by the client constitutes a full SMTP

mail that can be used for blacklist evaluation.

However, there is a small problem in this setup. The email that we see is slightly different than the email received on the server. This is because a mail server adds a **Received** header in the email after receiving the email. The received header contains the senders DNS name (or IP address) and the recipient DNS name (or IP address). In order to overcome this problem, we used the source IP address and the destination IP address to fake a **Received** header and added it to each email.

The emails are then fed to a spam detector and the sources in the legitimate received headers are consulted with the blacklists. A number of spam detectors can be used for our study. The two most popular and open source spam detectors are SpamAssassin [1] and DSpam [7]. DSpam requires manual training of individual mail boxes and so we used SpamAssassin in our experimental setup. SpamAssassin uses a number of spam detectors and assigns scores for each detector. The total score for a message is computed by adding the score of all detectors that classified the message as spam. If the total score exceeds the default threshold of 5.0, then the message is classified as spam. We used the default SpamAssassin configuration that came with the Gentoo Linux [3] distribution. We configured SpamAssassin with two additional detection modules namely Pyzor [8] and Razor [12] for improving SpamAssassin accuracy.

Blacklist lookups are done by reversing the IP addressing, appending the blacklist zone (eg, combined.njabl.org) and then making a DNS lookup. Remote DNS look ups cause significant latency, which makes evaluation on a large number of emails quite difficult. Therefore, we maintained a local copy of SORBS and NJABL and forwarded DNS queries for SpamHaus (Zen zone) blacklist to a local mirror. SpamCop queries were sent to the actual servers. We used BIND DNS server for these purposes and rbldnsd for serving local blacklists of SORBS and NJABL. The local copies of SORBS and NJABL were refreshed every 20 minutes.

SpamAssassin can itself be erroneous and so we need to first validate the usage of SpamAssassin as an oracle for spam detection. We do this by evaluating false positive and false negative of SpamAssassin on hand classified data sets of ham and spam.

4.2.2 Validating SpamAssassin

We evaluated SpamAssassin on email mailboxes that were hand classified into spam and ham. Table 4.1 shows four email accounts that we used for SpamAssassin evaluation. Account #1 contains

Spam-Assassin Threshold	Account #1 ham: 2,019 spam: 11,912		Account #2 ham: 5,547 spam: 107		Account #3 ham: 897 spam: 873		Account #4 ham: 4,588 spam: 482	
Threshold	FP	FN	FP	FN	FP	FN	FP	FN
4.0	1.14	4.17	0.25	3.08	0.89	3.67	0.76	5.39
4.5	0.84	4.47	0.02	3.08	0.56	3.78	0.61	5.60
5.0	0.45	4.88	0.02	4.02	0.56	4.24	0.50	5.60
5.5	0.30	5.80	0.02	4.02	0.45	5.27	0.22	6.22
6.0	0.25	6.06	0.02	4.02	0.33	6.41	0.11	6.85

Table 4.1: The false positive and false negative rates for SpamAssassin (at different thresholds) on four mail accounts that were manually sorted into spam and ham. Overall, SpamAssassin performs well.

all spam and ham collected in a work email account for over three years. Account #2 has been used for communicating with open source mailing lists. Account #3 belongs to a separate user who has used it for work and personal use. Account #4 belongs to another user who has used it for personal purposes for a number of years.

A message is a false positive for SpamAssassin if the message is ham and the SpamAssassin score for the message is greater than the given threshold. On the other hand, a message is a false negative for SpamAssassin if the message is spam and the SpamAssassin score is less than the threshold. The false positive rate is then computed as the ratio of false positives to the number of ham. The false negative rate is computed as the ratio of false negatives to the number of spam.

Table 4.1 shows the false positive rate and false negative rate of Spam Assassin on the four email accounts. We find that the false positive rate for SpamAssassin is very small and is close to 0.5% for a threshold of 5.0 (the default threshold in SpamAssassin). On the other hand, SpamAssassin has false negative rates of around 5%. Overall, SpamAssassin has very few false positive with manageable false negatives.

4.2.3 Evaluation

We deployed the entire system on an academic network for a period of around 10 days in June 2008. Figure 4.2 shows the number of mails per hour observed on the network. On an average, we observed $8,000$ SMTP connections per hour. However, half of these SMTP connections were aborted before the actual mail was transferred. This is because many mail servers in our network

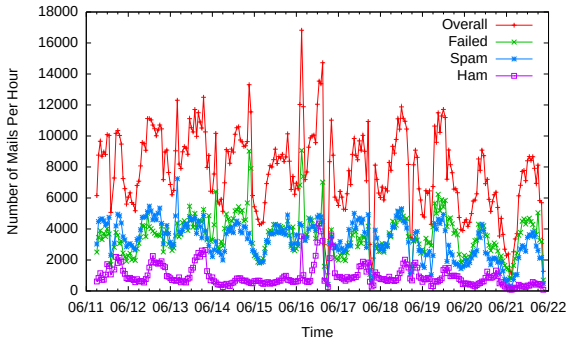


Figure 4.2: Number of mails per hour observed on the academic network. The overall mail rate is further divided by ham, spam, and failed connections.

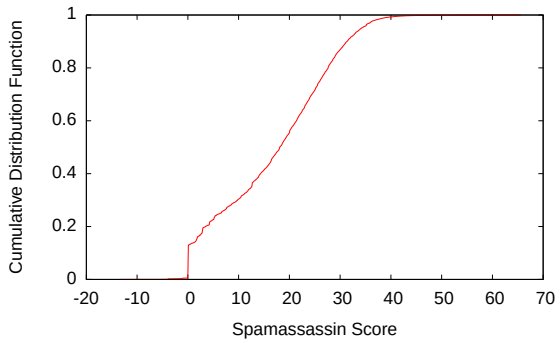


Figure 4.3: Cumulative distribution of SpamAssassin score for successfully delivered mail on the network (total = $1,074,508$).

were configured to reject a mail if the recipient was not a valid user in the domain. Spam and ham were separated using SpamAssassin and the rate of spam was significantly higher than the ham. In what follows we first present the characteristics of spam and ham observed on the network and then present the results on blacklist effectiveness.

4.2.3.1 Email Characteristics

Over the period of our experiment, we found that a total of $1,074,508$ emails were successfully delivered. Figure 4.3 shows the SpamAssassin score distribution for those mails. We find that roughly 15% of the mails received a score of 0 and around 20% of the mails were below the SpamAssassin threshold of 5.0. Over 70% of the mails received a score of more than 10.

Then we looked at the email sources and destinations. We observed a total of $53,579$ mail

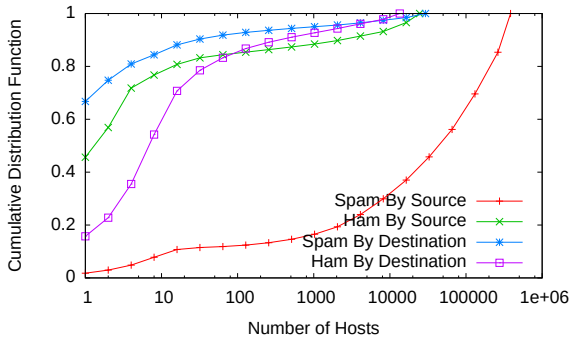


Figure 4.4: The source IP distribution and the destination IP distribution for spam and ham.

destinations with 64 of them within the academic network. Overall, we saw 609,199 mail sources with 111 within the academic network. Figure 4.4 shows the distribution of ham and spam by their sources and destinations. While spam was distributed across a large number of sources, the ham was concentrated to a very few sources. For example, while the top 10 hosts covered 80% of ham, the top 10 spamming sources covered less than 10% of spam. On the other hand the targets of spam were very concentrated when compared to ham. For example, while the top 10 destinations covered 80% of the spam, the top 10 destinations covered only 50% of ham. Overall, we find that the spam is well distributed across a large number of sources but targeted towards a few destinations. This is quite in contrast to the network level behavior of ham.

4.2.3.2 Blacklists Effectiveness

We now evaluate the false positive and false negative rates of four blacklists namely NJABL, SORBS (all zones), SpamCop (main zone) and SpamHaus (Zen zone). Table 4.2 shows the false positive rate of the four blacklists for different SpamAssassin thresholds. First, we find that the NJABL has the least false positives followed by SpamHaus. Second, the false positive rate of SpamCop and SpamHaus increases significantly when the SpamAssassin threshold is increased from 5.0 to 5.5. This indicates that the blacklists were positive for a number of messages that received the overall SpamAssassin score between 5.0 and 5.5. Finally, we look at unique source IPs for determining the false positive and false negative rates. We find that the false positive rates for unique source IPs are significantly higher when compared to the overall false positive rates. For example, SORBS has an overall false positive rate of 9.5%, but when unique source IPs are

SpamAssassin Threshold	NJABL		SORBS		SpamCop		SpamHaus	
	total	source IP	total	source IP	total	source IP	total	source IP
4.0	0.1	0.3	9.4	24.8	1.5	8.9	0.5	4.6
4.5	0.1	0.4	9.2	25.6	1.8	11.4	0.5	4.5
5.0	0.2	0.5	9.5	26.9	2.3	13.6	0.6	5.2
5.5	0.2	0.5	10.3	28.0	5.7	26.7	4.0	19.6
6.0	0.2	0.5	10.6	29.1	6.3	28.6	4.5	21.3

Table 4.2: False positive rate in percentage (overall and unique source IPs) for four different blacklists.

SpamAssassin Threshold	NJABL		SORBS		SpamCop		SpamHaus	
	total	source IP	total	source IP	total	source IP	total	source IP
4.0	98.4	98.1	65.4	59.2	36.4	40.4	38.0	41.4
4.5	98.4	98.1	64.9	59.2	35.4	40.3	36.9	41.2
5.0	98.4	98.1	64.8	59.2	34.9	40.2	36.3	41.0
5.5	98.4	98.1	64.5	59.1	34.7	40.2	36.2	41.0
6.0	98.4	98.1	64.4	59.1	34.5	40.1	35.9	40.8

Table 4.3: False negative rate in percentage (overall and unique source IPs) for the blacklists. Blacklists have a small false positive rate, but a large false negative rate.

considered the false positive rate increases to 26.9%. Overall, we find that SORBS has unreasonable amount of false positives but the other blacklists have few false positives.

Table 4.3 shows the false negative rates of the four blacklists for different SpamAssassin thresholds. While NJABL had a very few false positives, it has a huge false negative. For a threshold of 5.0 the false negative rate is 98.4%. SpamCop has the smallest false negative rate at around 36.3%. While the SpamAssassin threshold significantly impacted the false positive rate, its impact on the false negative rate is quite small. The false negative rates are around 59% for SORBS, 35% for SpamCop and 36% for SpamHaus. Overall the blacklists seem to have significantly higher false negative than we expected.

4.2.3.3 Exploring Overlap in Blacklists

In order to evaluate the coverage of different blacklists, we computed the number of times different blacklists agree on a spam. Figure 4.5 shows the percentage of spam detected by different blacklists and their mutual overlap. NJABL has been omitted because of its low detection rate. Surprisingly we find that the blacklists agree on a large number of spam. For example, SpamHaus

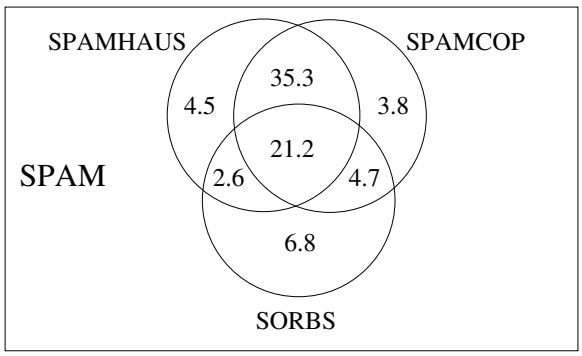


Figure 4.5: A venn diagram showing the overlap in blacklists for correctly flagged spam (overlap in true positives). There is a significant overlap among the blacklists.

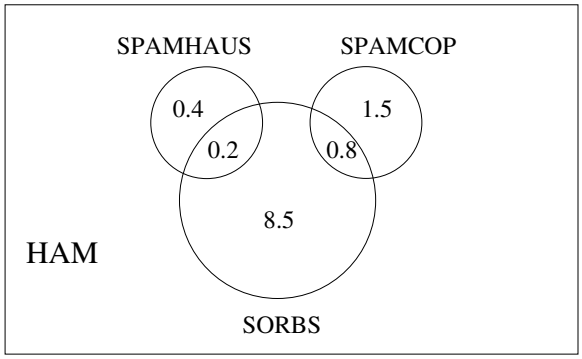


Figure 4.6: A venn diagram to show the overlap in blacklists in incorrectly flagging ham as spam (overlap in false positive). The blacklists rarely agree on these email messages.

and SpamCop agree on 57% of the spam, SORBS and SpamCop agree on 26% of the spam, and SORBS and SpamHaus agree on 24%. All three agree on 21% of the spam. The exclusive detection rate for the blacklists is small: 4.5% for SpamHaus, 3.8% for SpamCop and 6.8% for SORBS. Overall the four blacklists detect 79% of the spam. This implies that the spamtrap deployment for individual blacklists may overlap significantly and may not be diverse enough to capture the remaining 21% of the overall spam.

Figure 4.6 shows the overlap among blacklists for false positives with respect to SpamAssassin. We find that the blacklists disagree with each other on most false positives.

Top level domain	Emails received	% of total spamtrap mails	unique sources
.org	289,991	2.1	137,725
.org	449,803	3.2	216,291
.org	571,856	4.1	253,777
.com	1,090,611	7.8	407,838
.net	1,159,353	8.3	439,152
.net	1,306,411	9.4	473,686
.net	1,321,232	9.5	18
.com	1,458,865	10.5	486,675
.com	1,552,240	11.2	521,321
.net	1,698,295	12.2	513,057
.net	3,004,583	21.6	689,633

Table 4.4: Our spamtrap deployment by top level domains, number of emails received, and number of unique sources.

4.3 Exploring Inaccuracy of Current Techniques

We now embark on an exploration of the reasons for the false positives and false negatives we observed. Since neither the data nor the algorithms used to create the commercial blacklists are available, we begin by describing a new spamtrap deployment that allows us to look inside the blacklist generation processes at the root causes of the blacklist limitations. We then examine a variety of reasons for the false positives and false negatives observed including: target mail, low rate spam, detection delay, and a lack of local awareness.

4.3.1 New Spamtrap Deployment

Because the production blacklists do not provide any insight into their spamtrap deployment (e.g., spamtrap placement, number of spamtraps) or their techniques for translating mails received at their spamtraps into blacklist entries (e.g., thresholding), it is exceptionally difficult to examine the root causes of the false positives and false negatives they produce. As a result, we deployed our own spamtrap deployment covering 11 domains during the measurement period. The mail server in these domains copied mails sent to non-existent users to a separate account for post analysis. In total we observed $13,903,240$ emails from $1,919,911$ unique sources between February 10, 2009 to March 10, 2009. Table 4.4 shows the number of emails received and the number of unique sources observed on each of these domains. Over 14 million spam emails were captured and analyzed.

4.3.2 Fixing Inaccuracy in SpamAssassin

Earlier we found out that SpamAssassin had a false positive rate of less than 1% and a false negative rate of around 5%. It is important to understand the limitations of using SpamAssassin to evaluate blacklist accuracy so that we can guide our efforts of hand classification. A false negative for the SpamAssassin (i.e., a spam classified as ham) may appear to be a false positive for the blacklist if the blacklist is correctly pointing to it as spam. The false positive rates for our blacklist generation techniques in this paper are around 1% and the false negative rates of interest will be above 20%. Given the inaccuracy of SpamAssassin, the accuracy of false positives for the blacklist will be $FP_{blacklist} \pm FN_{spamassassin}$ or $1\% \pm 5\%$ and the accuracy of false negatives for the blacklist will be $FN_{blacklist} \pm FP_{spamassassin}$ or $> 20\% \pm 1\%$.

While the false positive rate of SpamAssassin is reasonable for evaluating the false negative rate of the blacklists, the false negatives of the SpamAssassin will clearly limit the conclusions in this paper. In order to overcome this problem, we hand classified the false negatives of the SpamAssassin. Instead of manually examining all false negatives of the SpamAssassin (potentially all legitimate mail), we only hand classified sources that hit spamtraps and also sent mails classified as ham by the SpamAssassin.

4.3.3 Mails on the Live Network

For these classes of experiments, we expanded our previous 10 day analysis to a month-long period from February 10, 2009 to March 10, 2009. Our month-long observation shows that roughly 75% of the delivered mail (i.e., ham and spam) was spam. Only 16% of the attempted connections (i.e., ham, spam, and failed connection) were legitimate mails. We observed 764,248 unique IP addresses during this period in 35,390 distinct BGP prefixes announced by 85 unique autonomous systems. Most of the spam messages (1,448,680) came from sources external to our network. However, we had a sizable number of spams (392,192) within the network, which was roughly four times the number of spam messages (98,679) from hosts within the network to the rest of the Internet (we send much more spam to ourselves than the rest of the Internet). Ham messages were dominated by internal to internal mails (369,431), followed by internal to external (151,860) and then by external to internal (114,792). The top five external senders (i.e., autonomous systems) of spam observed during this period at our network were Turk Telekom (69,278), Verizon (34,819), Telecomunica-

Number of domains	OR of domains		AND of domains	
	FP rate	FN rate	FP rate	FN rate
1	2.2	71.5	2.2	71.5
2	2.2	66.7	1.0	80.58
3	2.2	63.6	1.0	83.54
4	2.3	61.6	0.0	100.0
5	2.3	61.6	0.0	100.0
6	2.3	60.4	0.0	100.0
7	2.3	59.2	0.0	100.0
8	2.4	58.2	0.0	100.0
9	2.4	57.5	0.0	100.0
10	2.4	57.0	0.0	100.0
11	2.4	56.8	0.0	100.0

Table 4.5: The false positive and false negatives rates when the spamtrap deployment is expanded domain by domain.

coes Brazil (34,175), TELESC Brazil (27,360), and Comcast (25,576). The top five destinations (i.e., autonomous systems) for legitimate email from our network were Google (87,373), Inktomi (4,559), Microsoft (3,466), Inktomi-II (2,052), and Merit Networks (1,793). The average message size for all mails was 5,301 bytes, with averages of 4,555 bytes, 15,152 bytes, and 1,916 bytes for spam, ham, and failed connections respectively.

4.3.4 Causes of Inaccuracy

We now investigate four potential causes for the inaccuracy of current techniques by generating blacklists from our global spamtrap deployment and then applying them to the mails observed on the live network.

4.3.4.1 Targeted Mail

One possible explanation for the false negative rates observed by the blacklists is that some of the emails are part of a targeted spam campaign. Obviously, if a spammer sends targeted spam to a domain in which there are no spamtraps, it is impossible to blacklist the host. To explore the impact of this potential cause of false negatives, we examined the impact of spamtrap deployment size on accuracy. By building blacklists from spamtrap deployments of size 1, 2, ..., 11 we can explore the targeted nature of spam. Table 4.5 shows the result of this analysis. We consider two cases

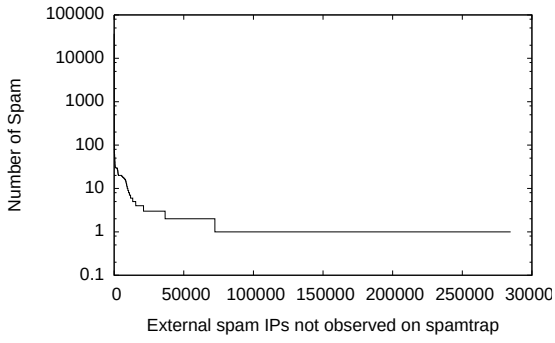
for blacklist generation, one in which an IP address is blacklisted if a spam host appears on any spamtrap domain, and one in which it is blacklisted if it appears on every spamtrap domain. The false negative rate for the OR of domains converge to roughly 56.8%, indicating that roughly 57% of the spam does not appear in any of the spam traps—a reasonable upper bound on the amount of targeted email. A lower bound on the amount of global mail can be seen in the false negative of rate the AND of domains, 100% after just three spamtrap domains are combined. Clearly, global spam seems to be quite limited. While a precise estimate is difficult without a universal deployment, it is clear that the blacklists are impacted by global and targeted behavior.

4.3.4.2 Low Volume Spam

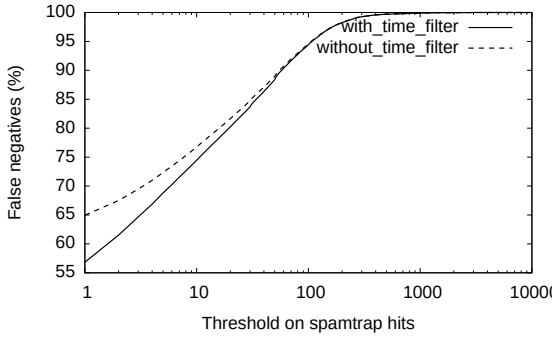
Another potential explanation of the false negatives observed is that although the campaigns are global, the vast number of hosts available to spammers makes it feasible to send a small handful of mails from each host to each target user or domain and still send millions of mails. This contributes to the problem of false negatives, in that most blacklist providers will not blacklist hosts for a single spam sent to a spamtrap. In order to investigate this phenomenon, we examined the spam sent to our network that was not observed on ANY of our spamtraps. For each spamming source, we calculated the number of spams sent to our network over the measurement period. As shown in Figure 4.7(a), while some spammers clearly sent numerous spams, the vast majority of sources sending spam to our network only sent a single spam. Therefore, any approach that requires multiple spamtrap hits will never report these high volume, single target sources as spammers.

4.3.4.3 Detection Delay

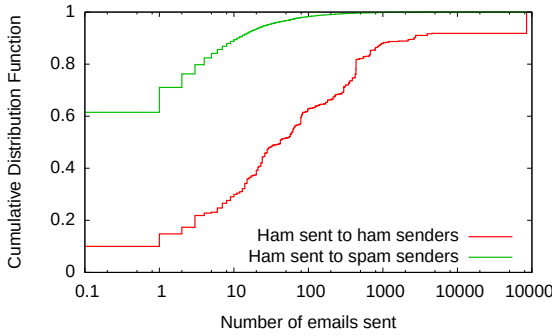
A third potential source of false negatives is the reactive nature of blacklist generation. By their nature, hosts are not put on blacklists until they send enough mail to spamtraps. During a fast global campaign, it is possible that we might receive the spam at the production network before it reaches a spamtrap or before the blacklist provider can send out an update. To explore the impact of this delay, we examined the idea of retroactive detection. That is, we created blacklists as expected, creating blacklist entries for spamming hosts only if they sent spam over a given threshold. We then enabled retroactive detection, that is, we classified hosts as spam if they sent mail to the spamtraps *at anytime during our observations* (potentially several weeks after we observed the spam). Figure 4.7(b)



(a)



(b)



(c)

Figure 4.7: (a) Number of mails sent by external spamming sources that were not observed on any spamtrap. Most of these sources sent just one spam to our network. (b) The difference in false negatives when blacklisting only spammers whose activity appears in a short window, versus those whose activity appear at anytime in the past. (c) The amount of legitimate mail sent by our network to networks that sent us spam and legimate mail.

shows the result of this analysis. For small threshold values (i.e., blacklisting when we see only one spam) the decrease in false negatives from retroactive detection is 10%. For higher thresholds, this value decreases. Thus 10% approximates a reasonable upper bound on the false negatives caused by delay.

4.3.4.4 Lack of Network Policy

False positives occur from blacklists when legitimate mail servers are blocked. Often times this occurs when a legitimate mail sever has been compromised or is being used by compromised hosts. In many cases this can be avoided, as the mail server can add the IP address of the sending host in the mail headers, but this is not always the case. For example, mails sent from Gmail web interface do not include the client's IP address, and as a result, blacklists are only left with the choice of blacklisting the server itself. What these blacklists lack is a notion of what servers are used and not used by a specific network. For example, consider the data in Figure 4.7(c). In this figure, we examine the amount of mail we sent to those networks that sent us spam and those that sent us ham. Note the stark contrast between the mail we sent to legitimate networks and those we send to spamming networks—90% of ham senders received more than one mail from us, while over 60% of spammers never received a single mail from our network. A few spamming domains received a large number of emails from us. As expected, these are false positives from web hosting sites as in the example above: Google (87,373), Inktomi (4,559), Microsoft (3,466). These sites could be whitelisted, but without knowing what services a network uses, this whitelisting may create false negatives. What blacklists need is a way to figure out what remote networks are important to a given network.

4.4 Context Aware Blacklist Generation

In the previous section, we saw that existing production reputation-based blacklists have a significant amount of false negatives and a non-trivial amount of false positives. We explored the reasons for these limitations through our own spamtrap deployment and discovered two broad classes of problems that hamper the creation of these lists: blacklists cannot block spam sources they have not seen (e.g., low volume, targeted, delayed) nor can they make decisions about which sources to blacklist unless they know if those sources are important to the networks. In this section, we describe

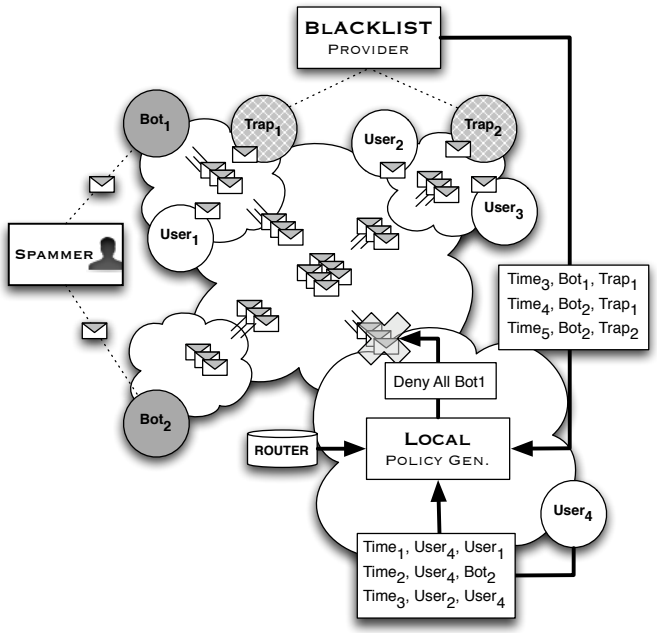


Figure 4.8: Our approach to reputation-based blacklist generation. Rather than enforcing a global "one size fits all" policy, a local generation algorithm combines local usage patterns (i.e, email usage), allocation information (i.e., BGP reachability), and global information (i.e., spamtraps) to make localized, tailored blacklist policies.

our approach to mitigating these limitations by incorporating the idea of local context. A representation of this approach can be seen in Figure 4.8. Rather than a "one size fits all" approach, which is embodied by the generation schemes for existing production blacklists and shown in Figure 4.1, our approach decides on blacklisting policy with the help of local information including usage patterns (i.e., email usage), network routing visibility (i.e., BGP information), as well as global information (i.e., spamtraps). With local context in hand, the policy generation mechanisms can eliminate false positives that occur from blacklisting locally important mail servers. In addition, the blacklisting can be more aggressive in blacklisting networks rather than individual sources—if these networks are not important in the local context. In this section we see how this general idea is applied in two specific improvements to blacklist generation: ratio-based blacklisting and speculative aggregation.

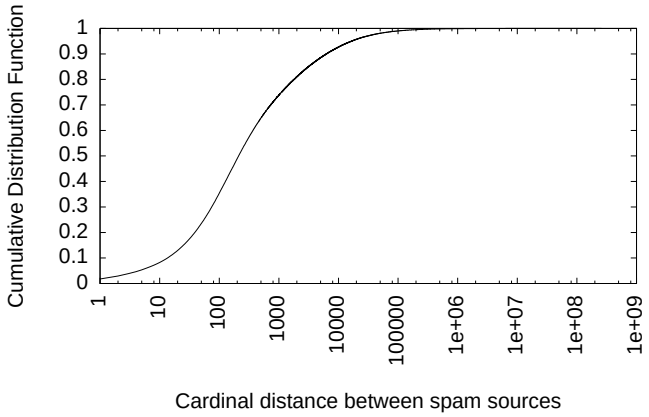


Figure 4.9: The cumulative distribution function for cardinal distance between spamming sources.

4.4.1 Ratio-based Blacklisting

In a simple threshold-based approach, a threshold is decided and an IP address is blacklisted if the number of mails sent to spamtraps crosses that threshold. However, the simple threshold mechanism can blacklist important mail servers (e.g., Gmail) if they are used to send even a small amount of spam. One solution to this problem is to compare local network traffic to the spamtrap emails. The assumption here is that a valid mail server will have significantly more mails delivered to valid addresses than to spamtraps while a spamming source will hit significantly more spamtraps than legitimate users in the live network. Therefore, we propose a ratio-based approach that computes the ratio of mails on the live network to the number of mails seen at the spamtrap and blacklists sources if the computed ratio is below a configured ratio. For example, consider that the configured ratio is 1 and a source IP is observed 5 times on the mail server and 10 times on the spamtrap. The ratio is $5/10 = 0.5$, which is lower than the provided ratio of 1, and will be blacklisted.

4.4.2 Speculative Aggregation

While a ratio-based approach addresses the false positive issues with the blacklists, the blacklists still exhibit a significant amount of false negatives. Recall from the previous section that this may be the result of low volume spammers, limited visibility, or detection delays. In each case, we can not blacklist that which we did not see or did not cross our threshold. In order to attack false positives resulting from sources we have not seen, the only solution we have is to speculate about

potentially bad sources. One potential source of information that we have to inform our prediction is the list of previous spamming sources. In Figure 4.9, we examine the impact of previous spamming sources to predicting future sources, by examining the distance of new spamming sources to existing spamming sources. Surprisingly, we note that most spamming sources are very close in IP address to the other spamming sources, most within a few hundred IP addresses of each other. Whether this is a result of dynamic address ranges or clustering of hosts into bad neighborhoods, we can make use of this technique to effectively predict spamming sources.

In order to detect these sources that do not hit spamtraps, we can use the context of local network traffic to determine bad and good neighborhoods with respect to the network. For example, consider a /24 network address range that has 75 active sources. However, if 50 of them are already blacklisted then it is quite likely that the remaining 25 are also spammers from the perspective of the network. So a heuristic that also looks at the bad neighborhood may be able to filter out these sources. For enlarging the scope of the blacklists we leveraged topological information available through Border Gateway Protocol (BGP). BGP is used to exchange routing information between autonomous systems (AS) and helps identify organizational and logical boundaries.

We aggregated traffic from spamtrap feeds and the live network by BGP prefixes and autonomous systems. Then we used three parameters for deciding to blacklist a network as oppossed to individual sources. First, the ratio of good to bad emails for the network is below the ratio provided in ratio-based approach. Second the ratio of bad to active sources in a network should be above a provided ratio. This parameter decides when we can speculatively classify an entire network as bad. However, we may blacklist networks even when we have seen very few sources from that network. Therefore the final parameter is the ratio of the minimum number of bad sources to total possible sources and, as such, increasing it would delay blacklisting a network.

4.4.3 Implementation

Having described our broad approach, we now provide the details on how mails from live networks and spamtraps will be aggregated, how blacklists will be generated and applied using our approach, and how entries will be removed from the blacklists.

4.4.3.1 Aggregating Sources in a Moving Time Window

The two streams of email messages, the spamtrap (bad events) and the live network mail (good events), are merged together using the timestamp on the emails. Sources are extracted and fed to the blacklist generation algorithms. We use a jumping window model to store network history. In this model, the events are stored for a given time window and the time window jumps periodically. For example, in a system with a history window size of 10 hours and a periodic jump of 15 minutes, the events are kept for 10 hours and the window jumps by 15 minutes. The counts in the last 15 minutes are then aged out. We accomplish this by keeping the count of each source in each jump window in a circular buffer and moving the current pointer to the circular buffer.

4.4.3.2 Generating Blacklists

For the threshold-based approach, we count the number of bad events (i.e., spamtrap hits) for each source IP and send the sources that cross a given threshold. For the ratio-based approach, we calculate the ratio of good events to bad events and send the sources for which the ratio is below a given ratio. The count for each address is taken over the history window. To enlarge blacklists from source IP address to BGP prefixes and autonomous systems, we take two additional parameters. A BGP prefix or an autonomous system is blacklisted if all three conditions are satisfied—the ratio of good events to bad events for the prefix is below the given ratio, the number of bad IP addresses to active addresses is above the minimum fraction, and the ratio of bad IP addresses to total possible addresses is above the specified threshold.

4.4.3.3 Applying Blacklists

The blacklists are generated periodically and the lists are refreshed each time. To save on messaging, the blacklist generation technique only emits new entries or instructions to remove old entries. These blacklists are then checked on the emails from live network until a new list is refreshed. In our implementation, we maintained the blacklist as a list of IP addresses in the open source database PostgreSql. We used the gist index ip4r for quickly checking whether a source IP is blacklisted.

4.4.3.4 Removal from Blacklists

Finally we need to lay down the policy for removing entries from the blacklist. For the threshold-based approach, an IP address is in blacklist until the network history has enough bad events from that IP address. When the network history for an IP address goes lower than the threshold, then the IP address is removed the blacklist. For the ratio-based approach, an IP address is removed from the blacklist when the ratio of good events to bad events goes above the specified ratio. BGP prefixes and autonomous systems are removed from blacklist if any of the three conditions fail—the ratio of good to bad events exceeds the specified ratio, or if the number of bad IP addresses to active IPs from the network falls below the provided threshold, or the ratio of bad IP addresses to total possible addresses falls below the threshold.

4.5 Evaluation

In this section, we evaluate the effectiveness of the existing techniques and compare it to the proposed techniques. In particular, we compare the three approaches to blacklist generation—the threshold-based approach, the ratio-based approach, and the speculative aggregation approach—by the false positive rate and the false negative rate. Since a bad human-chosen parameter can significantly impact blacklist generation, we compare the stability of the threshold-based approach with the ratio-based approach for a variety of chosen parameters. We further evaluate the impact of different parameters involved in speculative aggregation. Finally, because the blacklist sizes can increase over time, we compare the approaches by the CPU and memory requirements.

We used mails to spamtraps deployed in 11 domains as instances of bad events for generating blacklists. We used mails on a large academic network of 7,000 as instances of good activity as well for evaluating the effectiveness of blacklists. Both of these datasets were from February 10th, 2009 to March 10, 2009. We provided more details on the dataset and our experimental setup earlier.

For evaluating these techniques, we kept a history window of 10 hours and generated blacklists every 15 minutes. The history window jumped by 15 minutes after generating blacklists. The history window determines the time period for which source statistics are kept for generating the blacklists. Every 15 minutes the entries were added or removed from the blacklists and the new refreshed list was applied to the remaining network traffic. Each experimental run was performed

on roughly 2.5 million emails of live network traffic and 14 million emails on the spamtrap. Each run took almost one day on a machine with a Pentium Xeon 3.0 Ghz processor and was mostly I/O bound.

4.5.1 Comparing the Three Approaches

We now compare the simple threshold-based approach with the two approaches proposed in the paper: ratio-based approach and the speculative aggregation approach. Recall that in the threshold-based approach, an IP address is blacklisted if it has equal to or more spamtrap hits than the provided threshold. In the ratio-based approach, an IP address is blacklisted if the ratio of the number of good events (mails to the live network) to the number of bad events (mails to the spamtrap) is below the specified ratio. In the speculative aggregation approach, the IP addresses are aggregated by BGP prefixes and autonomous systems. Then BGP prefixes or autonomous systems are blacklisted instead of individual IP addresses if it is found that these networks are not of importance to one's network. Since the speculative approach uses a ratio-based technique for blacklisting individual IP addresses, it is essentially a combination of the ratio-based and speculation approaches.

Figure 4.10 shows the trade-off between the false negative rate and the false positive rate for the three approaches. First, we find that the ratio-based approach provides s significantly better false negative rate for any false positive rate provided by the threshold-based approach. Conversely, the ratio-based approach provides significantly better false positive rate for any false negative rate provided by threshold-based approach. For example, the false negative rate for the ratio-based approach is roughly 20% better than the threshold-based approach for false positive rates below 0.5%, which is roughly three times the detection rate of the threshold-based approach.

The speculation further improves detection rates over the ratio-based approach. The false negative rate improvement of the speculative approach over the threshold-based approach is between 30-40% for false positive rates below 0.5%, which is roughly 4-5x of the detection rate of the threshold-based approach. For false positive rates greater than 0.5%, the ratio-based approach provides slight improvement over threshold-based approach. Over this range, the speculative aggregation provides almost double the detection rate over the threshold-based approach.

The operational point for an approach is usually the knee in the false negative and false positive curve. For the threshold-based approach the knee is at 0.67% of false positive and 71% of false

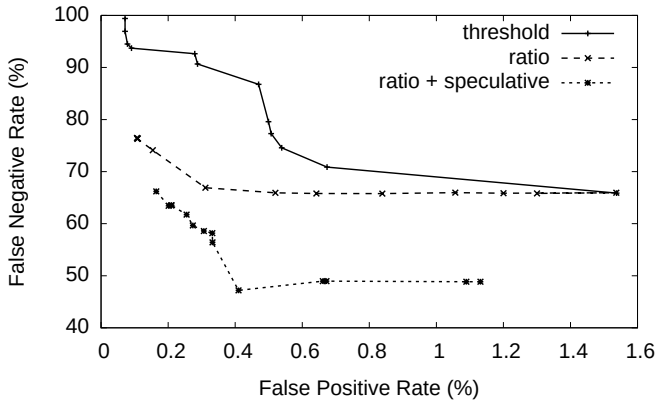


Figure 4.10: Trade-off curve of the false positive rate and the false negative rate for the three methods.

negatives and for the ratio-based approach the knee is at 0.31% of false positives and 67% of false negatives. For the speculative aggregation approach the knee is at 0.40% of false positives and 48% of false negatives.

4.5.2 Stability of the Threshold-based Versus the Ratio-based Approach

In both the threshold-based approach and the ratio-based approach, a network operator has to choose the threshold or the ratio for blacklisting. Since the thresholds are chosen by hand, we need to investigate how stable these schemes are for any given threshold. Table 4.6 shows the false positives and false negatives of the two approaches for different values of the thresholds and the ratios. For the threshold-based approach, the false positive rate increases suddenly from 0.67% to 1.54% when the threshold is reduced from 2 to 1. For the ratio-based approach, the increase in false positives is far more gradual. Looking at the data, we find that many mail servers in the network had one spamtrap hit in the time window of 10 hours.

4.5.3 Impact of Parameters on Speculative Aggregation

Recall that in speculative aggregation, BGP prefixes or autonomous systems are blacklisted if three conditions are satisfied. The first is if the ratio of good events (mails to the live network) to bad events (mails to the spamtraps) is below a specified ratio. The second is if the ratio of bad sources to total active sources is above a given threshold. Finally, is if the ratio of bad sources to total size

Threshold-based Approach			Ratio-based Approach		
Threshold	FP	FN	Ratio	FP	FN
1	1.54	65.9	100.000	1.30	65.8
2	0.67	70.9	75.000	1.20	65.8
3	0.54	74.6	50.000	1.05	65.8
4	0.51	77.3	25.000	0.83	65.8
5	0.50	79.6	10.000	0.64	65.8
10	0.47	86.8	5.000	0.52	65.9
15	0.29	90.7	1.000	0.31	66.9
20	0.28	92.6	0.010	0.15	74.1
25	0.09	93.7	0.005	0.11	76.3
30	0.08	96.9	0.001	0.09	76.4

Table 4.6: The values of the threshold-based and ratio-based approaches and the corresponding false positive and false negative rates.

of the BGP prefix or the autonomous system is above a given threshold.

Figure 4.11 shows the variation in the false positive rate and false negative rate for the speculative approach when the above three parameters are varied. The default ratio was kept at 0.1 and varied from 0.01 to 100. The ratio of bad IPs to total active sources was kept at 0.4 and varied from 0.1 to 0.99. The minimum ratio of bad IPs to total possible IPs in the network was kept at 0.01 and varied from 0.001 to 0.1. First, we find that the first and third parameters have significant impact on the false positive and false negative rates of the speculative aggregation approach. But varying the second parameter has very limited impact on the approach. Second, changing the minimum number of bad IP addresses provides a much better trade-off between false positive rate and false negative rate when compared to changing the ratio of good to bad events.

4.5.4 Performance

Figure 4.12(left) shows the growth of blacklists for the three techniques: threshold-based, ratio-based and speculative aggregation. We find that the growth of blacklist size is highest for the ratio-based techniques and lowest for the speculative-aggregation technique as it combines many sources into BGP prefixes. In order to see how blacklist size may impact the performance of the system, we created tables with different blacklist sizes in the database Postgresql (which is what we have used in our system). Then we created an index on the IP addresses and prefixes using GIST index in Postgresql. Table 4.12(right) shows the time to look up an entry and the index size for different sizes of the blacklist. We find that the time to look up an entry does not increase significantly, and

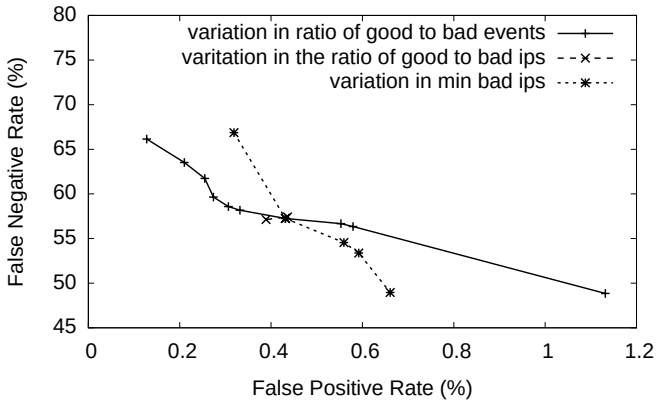
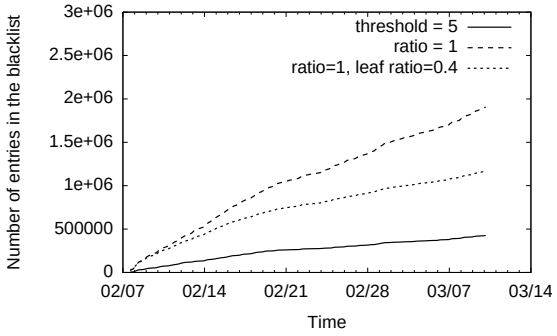


Figure 4.11: Impact of three parameters on the false positive rate and false negative rate for speculative aggregation.



Blacklist size	look up time (ms)	Index size
1000	0.045	40 KB
10,000	0.046	264 KB
100,000	0.050	3.7 MB
1 million	0.052	32 MB

Figure 4.12: (Left) The growth of the blacklist size for the three approaches. (Right) The impact of blacklist size on the time to look up an IP address. The index size grows linearly, but the lookup time for an entry is very fast.

for the month's operation, the index size is easily manageable.

4.6 Discussions

In this paper, we presented a detailed investigation of blacklist generation techniques using 2.5 million emails from a large academic network and 14 million emails from a spamtrap deployment in 11 domains. We first validated existing studies on effectiveness of commercial blacklists and observed that these blacklists have a significant amount of false negatives and a non-trivial amount of false positives. In order to better understand these issues and improve upon them, we presented a detailed analysis of ham and spam sources, based on our own spamtrap deployment. We then pro-

posed two improvements to the standard threshold-based blacklist approach. The first one reduces false positives by comparing traffic on the live network to the spamtrap hits for blacklisting sources. The second takes network traffic into account to safely aggregate bad sources into bad neighborhoods. The proposed techniques when combined together improved the false negative rate by 4-5x for false positive rates below 0.5% and 2x for false positive rates above 0.5%.

While effective at its goal of addressing the limitations of blacklist generation, this work has several limitations and opportunities for future work. First, the speculative aggregation technique presented in this paper is somewhat preemptive in nature. While our evaluation shows that the proposed technique provides significantly better trade-offs, it may be unacceptable to block traffic from hosts preemptively. Second, like other reputation-based systems, our blacklist generation system is also exposed to the attacks that increase or decrease the reputation of sources. While the ratio-based technique provides protection against attacks to blacklist a mail server, it is still vulnerable to attackers increasing the reputation of sources by sending a large number of mails to a legitimate user. Currently, our system only counts the total number of mails on the live network and is vulnerable to such an attack. A system that counts the number of unique users to which a source sends mail may be resilient to such an attack and will be explored in the future. Third, blacklist providers often indicate that they are not responsible for the blocking email as they only generate the blacklists, and it is the network administrators who are blocking the mails. However, these blacklists currently are generated centrally. The only option a network administrator has is to accept or reject a given blacklist. Our proposed deployment model requires either publication of raw spamtrap data to subscribers or the publication of (aggregate) local network traffic statistics to the blacklist providers. Finally, in our current implementation we only extracted the first "Received" header in the email messages. In our ratio-based mechanism, we could not blacklist sources if we did not blacklist the first source. In the future, we may like to add support for blacklisting of sources in received headers after the first one.

CHAPTER 5

Conclusions and Directions for Future Work

In the previous chapters we enhanced three network security systems by leveraging their deployment context. First, we demonstrated how the performance of a signature-based intrusion detection and prevention systems can be significantly improved by exploiting the rule-set and the network traffic characteristics. Then we showed how the configuration of a honeynet impacts its accuracy in threat detection and its ability to avoid fingerprinting. We demonstrated that a honeynet configured representative of the network will provide more accurate threat view on the network and will reduce its chances of getting fingerprinted. Finally, we introduced a blacklist generation system that leverages network traffic characteristics to significantly enhance blacklist accuracy.

5.1 Lessons Learned

While we investigated the impact of context information on three types of security systems, we did not produce a technique in the dissertation that can automatically adapt a new security system to the network context. Our general approach has been to identify contextual information that may be relevant to a security system and then rigorously analyze its impact on the accuracy and performance of the security system. Once we have identified the contextual information that is important, we developed techniques to automatically collect the information from the network and then modify the corresponding security system to incorporate the contextual information.

When we find a new security system, we would like to use our general approach to automatically adapt a security system to the network context. We now present a systematic way by which our general approach can be used to adapt a new security solution.

5.1.1 Determining the Contextual Information

Recall that the three broad types of security context in a network are the vulnerability profile, the attack surface, and the traffic usage model. The vulnerability profile relates to the applications and operating systems on a network. The attack surface relates to the observed attacks on a network, and the traffic usage model refers to how network-based applications are used on the network.

Determining the type of contextual information that is relevant and the granularity of such information is often the critical piece. For example, in our adaptation of intrusion detection system, we found that the traffic usage model and the security device configuration impacts the performance of the system. We aggregated this contextual information by the protocol fields and then used it for determining the rule evaluation strategy. Therefore, we used the contextual information at the granularity of protocol fields.

In honeynet configuration, we analyzed the attack surface as well as the vulnerability profile of a network. We then identified the vulnerability profile to be critical to honeynet accuracy. We used the vulnerability profile information at the granularity of operating system and application configuration for adapting the honeynet system.

For blacklist generation system, we found the traffic usage model to be relevant for accuracy. In particular, we found the mail delivery rate on a network for different IP addresses as the useful factor in blacklist accuracy.

So while determining the relevant contextual information, the following questions need to be answered:

- **Relevant contextual information:** Does a change in vulnerability profile, attack surface or traffic usage affect the accuracy or the performance of the security system? The information types that significantly impact the security system security system need to be considered for adaptation.

- **Granularity of information:** At what granularity does the relevant contextual information matter? The contextual information can then be aggregated at such granularity while adapting the security system to a new network.

5.1.2 Discovery of Contextual Information

Once the relevant contextual information is determined, we need to discover such information when deploying in a new network. To discover such information, the following questions need to be answered:

- **Manual discovery:** Is it difficult to manually discover? If so, we need to build automated tools for gleaning such information from a network.
- **Temporal variation:** Does the contextual information change with time? If so, the contextual information needs to be refreshed regularly and the security system to be accordingly updated.
- **Host modifications:** Does contextual information discovery require host modification? Usually network-based security systems do not require host modifications, but it may be required in certain systems that leverage host as well network views.
- **Active probing:** Does the discovery require active probing or passive monitoring? Passive monitoring should be preferred over active probing. However, if active probing is required, then it should be executed with care as it may impact the availability of the host and network resources.

5.1.3 Incorporating Context Into the Security System

The final step is to incorporate the contextual information available in the security system. The following questions may help one to decide how to adapt the system:

- **Design from scratch:** Are we designing a new system or modifying an existing system? Usually it is much easier to incorporate context in a system that is being designed from scratch, but a new system would also require more time to develop. So the amount of effort in modifying system should be estimated before designing a new system.
- **Cost of context awareness:** How much is the processor and memory overhead for context-aware changes? Usually adding more code for context awareness may require more CPU cycles and memory. So the potential cost should be benchmarked and compared with the potential benefit.

- **Mechanisms for reconfigurations:** Are the temporal context changes significant enough to cause regular updates to the system? If the answer is yes, then the system needs a mechanism for reconfiguring when the context changes.

5.2 Extending Context Aware Security To New Domains

Having described the broad approach for making a new security system context-aware, we now present two new types of security systems that may benefit from context adaptation.

- **Context-Aware Reputation-Based Systems:** Networks are threatened constantly with a large number of exploits and malware. The existing technologies for their detection like the intrusion detection systems and the anti-virus engines are constantly failing to come up with reliable signatures. We find that the detection rates for anti-virus engines have fallen below 60%. Because of the increasing difficulty in gaining visibility into current day attacks and developing reliable signatures, coarse-grained dynamic reputation-based systems are gaining ground.

  Reputation-based systems employ a wide variety of listing techniques including whitelists and blacklists of identifiable entities or behaviors. We believe a combination of such listing techniques will be a part of all security systems in future. However, for them to achieve the desired accuracy and performance, we believe that they will need to understand the context of their operation. We already demonstrated the utility of such context-awareness for one type of reputation-based systems, namely the spam blacklists. We now outline two types of reputation-based systems that will benefit by adapting to the network environment.

  - **Perimeter defense:** Enterprise networks are routinely facing attacks on their networks. On the other hand, there is a constant demand to open up the network for newer clients, customers, and applications. The traditional way of protecting the networks using static firewall rules is fast becoming un-manageable. One approach to resolve the tradeoff between functionality and security is to come up with dynamic generation of reputation-based whitelists and blacklists.

    The reputation-based system can aggregate suspicious traffic on the Internet and combine it with its local traffic characteristics to automatically generate whitelists and black-

lists. Such whitelists/blacklists will identify the regions of IP addresses and the services which are safe/dangerous from the perspective of a network.

- **Web application attacks:**

  Web applications have become very popular over time. However, these applications may have numerous vulnerabilities like remote code execution, cross-script scripting, and SQL injections that threaten the integrity of web service and the identity of web users. Web applications are inherently open but need to be protected against the unknown attacks.

  Often web application attacks are triggered by accessing a specific resource on the web application. For example, the CodeRed worm exploited the IIS server by accessing a .ida file on the server together with malicious argument. So if we can determine the part of the URL that is used for an attack, we can temporarily disable access to the resource to prevent the attack. We plan to aggregate legitimate usage on a network together with the suspicious traffic to determine reputation of different URLs. URLs are hierarchical in nature. We can aggregate the reputations in a hierarchy and then determine the broadest possible URLs that can be safely blocked and automatically stop the exploits.

- **Semantics-Aware Aggregates for Anomaly Detection Systems:** Netflow-based anomaly detection systems aggregate network traffic by different attributes like source IP, TCP destination port etc. Then these aggregates are monitored over time and an anomaly is flagged when the aggregate count changes significantly. For example, total bytes transferred can be aggregated by /24 networks, then byte counts are monitored for these aggregates and an anomaly is flagged when the count changes significantly. One of the problems is to determine which all aggregates to monitor. So traffic is aggregated at multiple granularities of an attribute in a hierarchy. For example, traffic can be aggregated at multiple granularities of source IP: individual IP addresses, /24 address space, /16 address space and /8 address space.

  However, such hierarchies are often statically and structurally decided. They do do not consider the semantic usage of the attribute and so the extracted aggregates do not reflect the semantic usage of the network. We plan to develop semantic hierarchies for network attributes like IP address and TCP ports. Aggregating traffic in these hierarchies will expose more accurate aggregates that better segment the network usage. For example, a semantic

hierarchy for IP address can be constructed by combining global routing data like BGP routes with the local routing data available via OSPF. Similarly, TCP ports can be aggregated into broad port classes like peer-to-peer applications, interactive applications, instant messengers etc. Then extract prominent aggregates, monitor the aggregates, and flag anomalies. We plan to evaluate the accuracy improvement in anomaly detection by using netflow data from a university network.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] The apache spamassassin project. http://spamassassin.apache.org/.

[2] Bleeding edge Snort. http://www.bleedingsnort.com/.

[3] Gentoo linux. http://www.gentoo.org/.

[4] Honeynet research alliance. http://honeynet.org/alliance/index.html.

[5] Nessus - network vulnerability scanner. http://www.dsbl.org.

[6] Not just another bogus list. http://njabl.org.

[7] Nuclear Elephant: The DSPAM Project. http://www.nuclearelephant.com.

[8] Pyzor. http://pyzor.sourceforge.net/.

[9] Sorbs DNSBL. http://www.sorbs.net.

[10] spamcop.net - beware of cheap imitations. http://www.spamcop.net/.

[11] The spamhaus project. http://www.spamhaus.org.

[12] Vipul's razor. http://razor.sourceforge.net/.

[13] Arbor Networks. Worldwide infrastructure security report, Sept. 2006.

[14] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The nepenthes platform: An efficient approach to collect malware. In *9th International Symposium On Recent Advances In Intrusion Detection*. Springer-Verlag, 2006.

[15] Michael Bailey, Evan Cooke, Tim Battles, and Danny McPherson. Tracking global threats with the Internet Motion Sensor. 32nd Meeting of the North American Network Operators Group, October 2004.

[16] Michael Bailey, Evan Cooke, Farnam Jahanian, Jose Nazario, and David 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.

[17] Michael Bailey, Evan Cooke, Farnam Jahanian, Niels Provos, Karl Rosaen, and David Watson. Data Reduction for the Scalable Automated Analysis of Distributed Darknet Traffic. *Proceedings of the USENIX/ACM Internet Measurement Conference*, October 2005.

[18] John Bethencourt, Jason Franklin, and Mary Vernon. Mapping Internet sensors with probe response attacks. In *Proceedings of the 14th USENIX Security Symposium*, Baltimore, MD, August 2005.

[19] J. Canavan. The evolution of malicious irc bots. In *Virus Bulletin Conference*, pages 104–114, 2005.

[20] Chinese Honeynet Team. Project status report: Period September 2005 to March 2006. `http://www.icst.pku.edu.cn/honeynetweb/honeyneten/statusreport_200604.htm`, March 2006.

[21] Young H. Cho and William H. Mangione. Programmable hardware for deep packet filtering on a large signature set. http://citeseer.ist.psu.edu/699471.html, 2004.

[22] William G. Cochran. *Sampling Techniques, 3rd Edition.* John Wiley, 1977.

[23] Evan Cooke, Michael Bailey, Farnam Jahanian, and Richard 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.

[24] Evan Cooke, Michael Bailey, Z. Morley Mao, David Watson, and Farnam Jahanian. Toward understanding distributed blackhole placement. In *Proceedings of the 2004 ACM Workshop on Rapid Malcode (WORM-04)*, New York, Oct 2004. ACM Press.

[25] Evan Cooke, Michael Bailey, David Watson, Farnam Jahanian Danny McPherson, and Z. Morley Mao. The Internet Motion Sensor Project. `http://ims.eecs.umich.edu`, June 2004.

[26] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstron, Lidong Zhou, Lintao Zhang, and Paul Barham. Vigilante: End-to-End containment of Internet worms. In *Proceedings of the ACM Symposium on Operating System Principles*, October 2005.

[27] Team Cymru. The darknet project. `http://www.cymru.com/Darknet/index.html`, June 2004.

[28] David Dagon, Xinzhou Qin, Guofei Gu, Julian Grizzard, John Levine, Wenke Lee, and Henry 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.

[29] Holger Dreger, Anja Feldmann, Vern Paxson, and Robin Sommer. Operational experiences with high-volume network intrusion detection. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 2–11, 2004.

[30] S. Egorov and G. Savchuk. SNORTRAN: An optimizing compiler for snort rules. Technical report, Fidelis Security Systems, 2002.

[31] U.M. Fayyad and K.B. Irani. What should be minimized in a decision tree? In *Proceedings of the Eighth National Conference on Artificial Intelligence*, 1990.

[32] Sheldon Finkelstein. Common expression analysis in database applications. In *Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, pages 235–245, New York, NY, USA, 1982.

[33] David Freedman, Robert Pisani, and Roger Purves. *Statistics*. Norton, 1998.

[34] French Honeynet Team. Status report: October 2005 – March 2006. , March 2006.

[35] Fyodor. Nmap. `http://www.insecure.org`.

[36] Georgia Tech Honeynet Team. Georgia tech honeynet report (September 15, 2005 – March 15, 2006). `http://www.ece.gatech.edu/research/labs/nsa/honeynet/status/2006-03/index.shtml`, March 2006.

[37] German Honeynet Team. Project status report: Period April 2005 to October 2005. `http://lufgi4.informatik.rwth-aachen.de/projects/honeynet/status2005-2`, October 2005.

[38] Google. Google safe browsing. `http://code.google.com/apis/safebrowsing/`, 2008.

[39] S. Graham, P. Kessler, and M. McKusick. gprof: A call graph execution profiler. In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction*, pages 120–126, June June, 1982.

[40] Dan Gusfield. *Algorithms on strings, trees, and sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.

[41] Xuxian Jiang and Dongyan 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.

[42] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM.

[43] Frank Knobbe. Wmf exploit. `http://www.securityfocus.com/archive/119/420727/30/60/threaded`, Dec., 2005.

[44] C. Kruegel and T. Toth. Automatic rule clustering for improved signature-based intrusion detection. Technical report, Distributed systems group: Technical Univ. Vienna, Austria, 2002.

[45] Christopher Kruegel, Fredrik Valeur, Giovanni Vigna, and Richard Kemmerer. Stateful intrusion detection for high-speed networks. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, page 285, Washington, DC, USA, 2002. IEEE Computer Society.

[46] Wenke Lee, João B. D. Cabrera, Ashley Thomas, Niranjan Balwalli, Sunmeet Saluja, and Yi Zhang. Performance adaptation in real-time intrusion detection systems. In *RAID*, 2002.

[47] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, K̃ristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, and Marc A. Zissman. Evaluating intrusion detection systems: The 1998 DARPA off-line

intrusion detection evaluation. In *Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX)*, pages 12–26, 2000.

[48] Microsoft. Microsoft security intelligence report: January-june 2006. http://www.microsoft.com/technet/security/default.mspx, October 2006.

[49] Microsoft. Vulnerability in graphics rendering engine could allow remote code execution. http://www.microsoft.com/technet/security/bulletin/ms06-001.mspx, January, 2006.

[50] MIT Lincoln Laboratory. 1998/1999 DARPA off-line intrusion detection. http://www.ll.mit.edu/SST/ideval/, 1999.

[51] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage. Network telescopes. Technical Report CS2004-0795, UC San Diego, July 2004.

[52] G.K. Mostefaoui, J. Pasquier-Rocha, and P. Brezillon. Context-aware computing: a guide for the pervasive computing community. *Pervasive Services, 2004. ICPS 2004. Proceedings. The IEEE/ACS International Conference on*, pages 39–48, 19-23 July 2004.

[53] George C. Necula, Scott Mcpeak, and Westley Weimer. CCured: type-safe retrofitting of legacy code. In *Principles of Programming Languages (POPL)*, pages 128–139, Portland, OR, January 2002.

[54] James Newsome, David Brumley, and Dawn Xiaodong Song. Vulnerability-specific execution filtering for exploit prevention on commodity software. In *Network and Distributed System Security Symposium*, 2006.

[55] James Newsome and Dawn Xiaodong Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Network and Distributed System Security Symposium*, 2005.

[56] Marc Norton and Daniel Roelker. SNORT 2.0: Hi-performance multi-rule inspection engine. Technical report, Sourcefire Inc., 2002.

[57] Norwegin Honeynet Team. Bi-annual status report 2005 q4. http://www.honeynor.no/docs/Honeynor_bi-annual_statusreport_2005q4.pdf, 2005.

[58] Jon Oberheide, Evan Cooke, and Farnam Jahanian. Cloudav: N-version antivirus in the network cloud. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, San Jose, CA, July 2008.

[59] Jooseok Park and Arie Segev. Using common subexpressions to optimize multiple queries. In *Proceedings of the Fourth International Conference on Data Engineering*, pages 311–319, Washington, DC, USA, 1988. IEEE Computer Society.

[60] Vern Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[61] Honeynet Project. Know Your Enemy: Learning with VMware. 2003.

[62] Niels Provos. Honeyd — A virtual honeypot daemon. In *10th DFN-CERT Workshop*, Hamburg, Germany, February 2003.

[63] Niels Provos. A Virtual Honeypot Framework. In *Proceedings of the 13th USENIX Security Symposium*, pages 1–14, San Diego, CA, USA, August 2004.

[64] Moheeb Abu Rajab, Fabian Monrose, and Andreas Terzis. Fast and evasive attacks: Highlighting the challenges ahead. In *Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, Sept. 2006.

[65] Anirudh Ramachandran, Nick Feamster, and Santosh Vempala. Filtering spam with behavioral blacklisting. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 342–351, New York, NY, USA, 2007. ACM.

[66] Eric Rescorla. Security holes... Who cares? In *Proceedings of USENIX Security Symposium*, August 2003.

[67] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of Usenix Lisa Conference*, November, 2001.

[68] David Schuehler and John Lockwood. A modular system for FPGA-based TCP flow processing in high-speed networks. In *14th International Conference on Field Programmable Logic and Applications (FPL)*, pages 301–310, Antwerp, Belgium, August 2004.

[69] R. Sekar, Y. Guang, S. Verma, and T. Shanbhag. A high-performance network intrusion detection system. In *ACM Conference on Computer and Communications Security*, pages 8–17, 1999.

[70] T. Sellis and S. Ghosh. On the multiple-query optimization problem. *IEEE Transactions on Knowledge and Data Engineering*, 2(2):262–266, 1990.

[71] Timos K. Sellis. Multiple-query optimization. *ACM Trans. Database Syst.*, 13(1):23–52, 1988.

[72] Sushant Sinha, Michael Bailey, and Farnam Jahanian. Shedding light on the configuration of dark addresses. In *Proceedings of Network and Distributed System Security Symposium (NDSS '07)*, February 2007.

[73] Sushant Sinha, Michael Bailey, and Farnam Jahanian. Shades Of Grey: On the effectiveness of reputation based blacklists. In *International Conference on Malicious and Unwanted Software (Malware 2008)*, October 2008.

[74] Andrew B. Smith and Jason M. Fox. RandomNet. `http://www.citi.umich.edu/u/provos/honeyd/ch01-results/3/`, March 2003.

[75] Robin Sommer and Vern Paxson. Enhancing byte-level network intrusion detection signatures with context. In Vijay Atluri and Peng Liu, editors, *Proceedings of the 10th ACM Conference on Computer and Communication Security (CCS-03)*, pages 262–271, New York, October 27–30 2003. ACM Press.

[76] Lance Spitzner. *Honeypots: Tracking Hackers*. Addison-Wesley, 2002.

[77] Klaus Steding-Jessen. The honeynet project: Distributed honeynet deployment in Brazil. http://www.honeynet.org.br/presentations/hnbr-dod-honeynet-project2006.pdf, 2006.

[78] Symantec. Symantec Internet threat report: Trends for July '05 - December '05. http://www.symantec.com/enterprise/threatreport/index.jsp, March, 2006.

[79] Symantec Corporation. DeepSight Analyzer. http://analyzer.securityfocus.com/, 2005.

[80] UK Honeynet Team. Status report for the period October 2005 – March 2006. http://www.ukhoneynet.org/reports.html, March 2006.

[81] Johannes Ullrich. DShield. http://www.dshield.org, 2000.

[82] Unspam Technologies. Project honey pot. http://projecthoneypot.org, 2008.

[83] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, fidelity and containment in the Potemkin virtual honeyfarm. In *Proceedings of the 20th ACM Symposium on Operating System Principles (SOSP)*, Brighton, UK, October 2005.

[84] Tim Weber. Criminals may overwhelm the web. http://news.bbc.co.uk/1/hi/business/6298641.stm, January 2007.

[85] Rafal Wojtczuk. libnids, June 2004.

[86] S. Wu and U. Manber. A fast algorithm for multi-pattern searching. Technical report, Department of Computer Science, University of Arizona, 1993.

[87] Yinglian Xie, Fang Yu, Kannan Achan, Eliot Gillum, Moises Goldszmidt, and Ted Wobber. How dynamic are IP addresses? In *SIGCOMM '07: Conference on Applications, technologies, architectures, and protocols for computer communications*, pages 301–312, New York, USA, 2007.

[88] Vinod Yegneswaran, Paul Barford, and Dave 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.

[89] Jian Zhang, Phillip Porras, and Johannes Ullrich. Highly predictive blacklisting. In *Usenix Security Symposium*, 2008.