Real-Time Detection of Malicious Network Activity Using Stochastic Models

by

Jaeyeon Jung

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2006

Author . . . . . .
Department of Electrical Engineering and Computer Science
May 22, 2006

Certified by . . . . . .
Hari Balakrishnan
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

To my princess Leia

Real-Time Detection of Malicious Network Activity Using Stochastic Models

by
Jaeyeon Jung

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2006, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

This dissertation develops approaches to rapidly detect malicious network traffic including packets sent by portscanners and network worms. The main hypothesis is that stochastic models capturing a host's particular connection-level behavior provide a good foundation for identifying malicious network activity in real-time. Using the models, the dissertation shows that a detection problem can be formulated as one of observing a particular "trajectory" of arriving packets and inferring from it the most likely classification for the given host's behavior. This stochastic approach enables us not only to estimate an algorithm's performance based on the measurable statistics of a host's traffic but also to balance the goals of promptness and accuracy in detecting malicious network activity.

This dissertation presents three detection algorithms based on Wald's mathematical framework of sequential analysis. First, *Threshold Random Walk* (TRW) rapidly detects remote hosts performing a portscan to a target network. TRW is motivated by the empirically observed disparity between the frequency with which connections to newly visited local addresses are successful for benign hosts vs. for portscanners. Second, it presents a hybrid approach that accurately detects scanning worm infections quickly after the infected local host begins to engage in worm propagation. Finally, it presents a targeting worm detection algorithm, *Rate-Based Sequential Hypothesis Testing* (RBS), that promptly identifies high-fan-out behavior by hosts (e.g., targeting worms) based on the rate at which the hosts initiate connections to new destinations. RBS is built on an empirically-driven probability model that captures benign network characteristics. It then presents RBS+TRW, a unified framework for detecting fast-propagating worms independently of their target discovery strategy.

All these schemes have been implemented and evaluated using real packet traces collected from multiple network vantage points.

Thesis Supervisor: Hari Balakrishnan
Title: Professor of Computer Science and Engineering

Acknowledgments

I am indebted to Hari Balakrishnan for giving me invaluable advice on countless occasions not only on research but also on various issues of life. Hari is living evidence that even a man of genius can be warm, kind and always encouraging.

The summer of 2003 at the ICSI Center for Internet Research (ICIR) was one of the most productive periods in my six years of Ph.D study. I am deeply grateful to Vern Paxson for his brilliant insights on data analysis, thoughtful discussions, and detailed comments on my dissertation. I would also like to thank Frans Kaashoek and Robert Morris for their advice on the dissertation.

This dissertation would not have been possible without many collaborative efforts with Arthur Berger, Rodolfo Milito, Vern Paxson, Stuart Schechter, and Hari Balakrishhan. They demonstrated hard work, clear thinking, and remarkable writing skills that I learned through collaboration with them. I am also grateful to Emre Koksal for answering my tedious questions on stochastic processes.

Kilnam Chon, my advisor at the Korea Advanced Institute of Science and Technology (KAIST) taught me never to be afraid of a challenge and always to tackle a hard problem. During my internship at the Cooperative Association of Internet Data Analysis (CAIDA), K Claffy showed an incredible passion for research. Evi Nemeth was always enthusiastic about teaching and caring for students. I would like to thank all of them for being great mentors.

It has been a pleasure to get to know and to work with so many wonderful colleagues at the Networks and Mobile Systems (NMS) group. Thanks to Magdalena Balazinska for her sense of humor and friendship; Michel Goraczko for his help on any computer and program related problems; Nick Feamster for collaboration and numerous discussions; Dave Andersen for his encyclopedic knowledge and sharing cute random Web pages; Allen Miu and Godfrey Tan for fun lunch chats. There are also many people on the G9 that helped me go through many long days. Thanks to Athicha Muthitacharoen, Jinyang Li, Emil Sit, Michael Walfish, and Mythili Vutukuru. Special thanks to Sheila Marian for her tremendous help in dealing with administrative work, for throwing NMS parties and for squeezing me into always-overbooked Hari's schedule.

I have been delighted to have many Korean friends at MIT. In particular, I would like to thank some of them who came to my defense talk to show their support: Youngsu Lee, Alice Oh, Daihyun Lim, and Jaewook Lee. I am eternally grateful to Yunsung Kim for being my best friend in high school, in college and at MIT.

I cannot thank my parents, Joonghee Jung and Hanjae Lee, enough for their endless love, support, and encouragement. I am a proud daughter of such awesome parents and hope that my daughter, Leia, will feel the same way when she grows up. I would also like to thank my brother Jaewoong and parents-in-law, Judy & Lowell Schechter for their support. Finally, I am especially grateful to my husband, Stuart, who is also my scuba buddy, crew mate, and research collaborator.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A network worm automatically spreads from computer to computer by exploiting a software vulnerability that allows an arbitrary program to be executed without proper authorization. One experimental study reports that the Sasser worm located a PC running a vulnerable operating system and successfully compromised the machine in less than four minutes from when the machine was connected to the Internet [78]. Another example that is frequently cited is the Slammer worm, which broke into the majority of vulnerable hosts on the Internet in less than ten minutes, congested many networks and left at least 75,000 hosts infected [44].

Once compromised, a host can be used for such nefarious activities as launching Denial-of-Service (DoS) attacks, relaying spam email, and initiating the propagation of new worms. For instance, the Bagle worm downloads a program that turns an infected machine into a *zombie* that an attacker can control remotely [15]. As a result, a large number of exploitable machines are readily available to an attacker, thus facilitating distributed and large-scale automated attacks. Furthermore, because attackers tend to exploit a vulnerability soon after it becomes known[1] it is extremely difficult for system administrators to patch every vulnerability on their machines before a new malicious code is released.

The speed and prevalence of automated attacks render ineffective any legacy defenses that rely on the manual inspection of each case. It is necessary to deploy an automated defense system that continuously monitors network traffic, determines whether the traffic from a particular source reveals a certain malicious activity, and then triggers an alarm when it finds such traffic. In this dissertation, we investigate the problem of designing efficient detection mechanisms for malicious network activity that are suitable for practical deployment in a real-time automated network defense system.

Our thesis is that *stochastic models that capture a host's particular connection-level activity are a good basis for the detection algorithms that we seek.* A host generates packets (or connections) in order to communicate with certain applications running on a remote host. The host's network activity can be observed by a traffic monitor that is placed on a network

[1] Symantec reports that it took 5.8 days on average between the public disclosure of a vulnerability and the release of an associated exploit during the first six months of 2004 [70].

pathway where those packets pass by. Using such a monitor, we can characterize the host's patterns of network access based on the metrics such as a connection initiating rate and a connection failure ratio. In principle, these patterns reflect a host's activity (i.e., benign Web browsing vs. scanning), and we construct models of what constitutes malicious (or benign) traffic and develop online detection problems that promptly identify malicious network activity with high accuracy.

The rest of this chapter gives an overview of various malicious network activities intended to break into a host or to disrupt the service of a variety of network systems; examines the challenges to detecting such attacks in real-time; reviews previous detection schemes; and discusses some of the key contributions of this dissertation.

1.1 Malicious Network Activity

Many Internet attacks involve various network activities that enable remote attackers to locate a vulnerable server, to gain unauthorized access to a target host, or to disrupt the service of a target system. In this section, we survey three common malicious activities: vulnerability scanning, host break-in attack, and DoS attack. These three malicious activities are only a subset of many Internet attacks occurring lately, but their prevalence has drawn much attention from the research community.

1.1.1 Vulnerability Scanning

There are many tools that allow network administrators to scan a range of the IP address space in order to find active hosts and to profile the programs running on each host to identify vulnerabilities [2,3,4]. These scanning tools, when used by an attacker, can reveal security holes at a site that can then be exploited by subsequent intrusion attacks. One such tool is "nmap": Figure 1-1 shows that `nmap` correctly identifies two open ports and the operating system that `tennis.lcs.mit.edu` is running.

```
Starting nmap 3.70 ( http://www.insecure.org/nmap/ )
Interesting ports on tennis.lcs.mit.edu (18.31.0.86):
PORT   STATE SERVICE
22/tcp open  ssh
80/tcp open  http
Device type: general purpose
Running: Linux 2.4.X|2.5.X
```

Figure 1-1: The `nmap` report for `tennis.lcs.mit.edu`

1.1.2 Host Break-in Attack

When a target machine is located, an attacker attempts to break in to the system in order to gain unauthorized use of or access to it. The most common method of intrusion is to exploit

a vulnerability of one of the servers that the target machine is running. According to the US-CERT vulnerability notes database [77], there have been more than 1,300 vulnerabilities found since 2001, among which over 7% are considered serious and announced as US-CERT technical alerts.[2] Table 1.1 shows annual vulnerability statistics compiled from the US-CERT database [77].

Table 1.1: Vulnerabilities listed in the US-CERT database as of July 26, 2005

Year	Vulnerabilities Found	Vulnerabilities with a Severity Metric > 40
2005	151	4
2004	345	13
2003	225	22
2002	303	27
2001	327	32
2000	90	4
$<$ 2000	52	9

An attacker sometimes directly aims at an authentication system. Despite its known weaknesses [47,49], password-based authentication is still popular among Web sites and SSH [92] servers. When presented with a correct pair of a username and a password, a system grants access to anyone — to break-in, an attacker must guess a matching pair of a username and a password, which are each usually composed of several alphanumeric characters. There are cracking tools [50,64] available that reduce the time required to guess passwords using pre-built word-lists.

Network worms and viruses automate host break-in attacks and propagate to other hosts on the Internet. Section 3.2 provides an in-depth discussion of network virus and worm propagation.

1.1.3 Denial-of-Service Attack

A Denial-of-Service (DoS) attack is an attack in which an attacker floods a target system with malicious traffic in order to prevent legitimate access. DoS attacks can both overload the victim host or its Internet link and causes a partial or complete service disruption [43, 74].

The impact is more severe when an attacker launches coordinated DoS attacks by thousands of machines, termed distributed DoS, or DDoS attacks [18]. Anecdotal evidence suggests that a *botnet*, a group of compromised machines controlled over an IRC channel or some other mechanism, is frequently used for DDoS attacks [51]. For more discussion of DoS attacks and defense mechanisms, one can refer to the book by Mirkovic *et al.* [43].

[2] US-CERT assigns a severity metric to each vulnerability, ranging from 0 to 180. Vulnerabilities with a metric greater than 40 are candidates for US-CERT alerts.

1.2 Challenges to Real-Time Detection

To cope with the increasing threat of various Internet attacks, many networks employ sensors that monitor traffic and detect suspicious network activity that precedes or accompanies an attack. For a network of hundreds (or thousands) of machines, having such sensors is indispensable because it is a daunting task to ensure that every machine is safe from known vulnerabilities, especially when machines are running different operating systems or network applications.

In this section, we discuss the technical challenges in developing practical and deployable algorithms that detect malicious network activity in real-time. Each challenge sets up a goal that we intend to meet when designing a detection algorithm. We briefly describe our general approach to meeting each goal, leaving the detailed discussions for the remaining chapters.

1.2.1 Detection Speed and Accuracy

Many Internet attacks are composed of multiple instances of malicious payload transmissions. For instance, scanning activity generates a sequence of probe packets to a set of destinations, and a host infected by a computer worm or virus spreads an exploit to many other vulnerable hosts. Typically, gathering more data improves accuracy as individual pieces of evidence collectively give greater confidence in deciding a host's intent. A host that attempted to access only one non-existent host may have done so because of misconfiguration rather than scanning activity. In contrast, a host that continued to send packets to dozens of different non-existent hosts may well be engaged in scanning activity. An important question that we aim to answer in this dissertation is the following: **how much data is enough to determine a host's behavior with a high level of confidence?**

Fast detection is one of the most important requirements of a real-time detection system along with high accuracy. The early detection of a port scanner gives network administrators a better chance to block the scanner before it launches an attack: if a scanner has probed 25 distinct machines and found 1 vulnerable host that it can exploit, then it may send out an attack payload immediately. If a scan detection algorithm had been based on the connection count with a threshold of 20, it could have blocked this attack, but if the algorithm had a threshold of 30, it could not have. In principle, as more observations are needed to make a decision, more malicious traffic will be allowed to pass through the system.

The trade-off between accuracy and detection speed complicates the design of a real-time detection algorithm, and we seek to provide a quantitative framework that allows us to gauge an algorithm's performance. To this end, we design detection algorithms using the *sequential hypothesis testing* technique [80], which provides a mathematical model for reducing the number of required observations while meeting the desired accuracy criteria. Chapters 4, 5, and 6 present the algorithms developed based on this framework and show that those detection algorithms significantly speed up the decision process.

1.2.2 Evasion

Attackers can craft packets and adjust their behavior in order to evade a detection algorithm once the algorithm is known. This never-ending arms race between attackers and defense systems requires that a detection algorithm be resilient to evasion to the possible extent. One way to achieve a high resiliency is to define strict rules for permissible activity such that only well-defined network traffic goes through the system. Even so, an attacker may find a way to alter its traffic pattern to appear benign. But, such modification often comes at the cost of a significant drop in attack efficiency.

In any case, it is important to precisely define a model for malice and take into consideration possible variants and exceptional cases that the model may not cover. One possibility is to build a stronger defense system by layering multiple independent detection algorithms. For each algorithm that we develop, we discuss the algorithm's limitations in detail and describe what should supplement the algorithm in order to raise the bar.

1.2.3 False Alarms

There are three main reasons for false alarms. First, the model of malicious activity based on which a detection algorithm operates may include uncommon benign activities. Because of the high variation in traffic patterns generated by each application, it is difficult to factor in all the possible cases leading to a false alarm. Hence, it is important to test a detection algorithm over *real* network trace data that include different types of traffic, and to refine the model such that it eliminates identified false alarms.

Second, the same activity can be regarded as permissible or prohibited depending on a site's security policy. For example, many peer-to-peer client applications scan other neighboring hosts to find the best peer from which to download a file. This scanning activity can exhibit a similar traffic pattern to that of a local host engaged in scanning worm propagation. Depending on a site's policy on allowing peer-to-peer traffic, flagging a peer-to-peer client as a scanner can be a legitimate detection or a false alarm. We consider this policy issue in evaluating the performance of our detection algorithm, separating obvious false alarms from policy-dependent ones.

Third, an attacker can spoof an originating IP address so that a detection system erroneously flags an otherwise innocent host. This framing attack can cause a "crying wolf" problem: a real attack would not be taken seriously should an immediate response take place. While our detection algorithm operates based on the assumption that each source IP address is genuine, a simple additional component can prevent framing attacks (see Sections 4.4 and 5.5 for details).

1.2.4 Scaling Issues

The amount of network traffic determines the amount of input data that a detection system needs to analyze in real-time. A detection system employed at a busy network faces the challenge of processing data at high rates. First, the high volume of input data requires a high degree of accuracy. If a detector is designed such that its false alarm rate is 0.001,

the actual number of false alarms is that rate multiplied by the number of events generated by input traffic, which increases proportionally with the amount of traffic arriving at the network monitor.

Second, many operational issues need to be addressed for running a real-time detection system in high-volume networks. As discussed by Dreger *et al.* [19], factors that impact the system's CPU load and memory usage include the type of analysis (e.g., transport-layer analysis vs. application-layer analysis), the rate of packet arrivals to the detection system, and the amount of connection state to be maintained in memory. Hence, it is crucial to have efficient resource management schemes in place in order to keep the detection system operational as traffic volume rises.

Third, real-time detection algorithms must be efficient enough to not overload the system when the state it maintains grows with respect to traffic. Moreover, an attacker can also target the detection system itself if he knows how to slow it down by consuming too many resources [52, 54]. The detection system should be aware of how fast the state can grow with respect to the traffic volume and should provide guards that limit its resource usage.

1.3 Detection Schemes

There is a large body of literature that presents, evaluates, and surveys detection approaches of network intrusion attacks. These approaches differ depending on their goal of detection and set of rules required for operation, but, we can roughly categorize them based on the underlying detection principle. In this section, we discuss three major detection principles with emphasis on their strength and weakness.

1.3.1 Misuse Detection

Misuse detection (or signature-based detection) tries to detect known attacks when they occur [34,39,41,56]. It uses predefined attack signatures and compares a current event against these signatures. Figure 1-2 shows a Snort's rule for detecting a remote host attempting to install the SubSeven trojan horse program [32] on a local host.

```
alert tcp $EXTERNAL_NET 27374 -> $HOME_NET any
(msg:"BACKDOOR subseven 22"; flow:to_server,established;
content:"|0D 0A|[RPL]002|0D 0A|"; reference:arachnids,485;
reference:url,www.hackfix.org/subseven/;
classtype:misc-activity; sid:103; rev:7;)
```

Figure 1-2: Snort rule 103 for the SubSeven trojan horse: This rule triggers an alarm if Snort sees an incoming connection from port 27374 and the connection's payload contains the pattern specified in the "content:" field.

Because each detection results from a known attack, it is straightforward to diagnose the alarms generated by misuse detection systems. But, misuse detection is ineffective against novel attacks or slightly modified attacks whose signature is not available.

1.3.2 Anomaly Detection

Anomaly detection is designed to identify a source exhibiting a behavior deviating from that normally observed in a system [7, 31, 42]. The construction of such a detector begins with developing a model for "normal behavior". A detection system can learn the normal behavior by training data sets collected over a certain time period with no intrusions. It can also use for detection a set of rules describing acceptable behavior that an operator manually programmed in.

Anomaly detection is capable of detecting previously unknown attacks so long as those attacks form a poor match against a norm. But, it often suffers from a high degree of false alarms generated by legitimate but previously unseen behavior.

1.3.3 Specification-based Detection

Specification-based detection watches for a source that violates the pre-built program specifications [38, 58]. In particular, programs that run with a high privilege (e.g., SUID root programs [30]) need a careful watch since an attacker often targets such programs in order to execute an arbitrary code.

Like anomaly detection, specification-based detection can detect novel attacks, but the development of detailed specifications requires thorough understanding of many security-critical programs in a system, which can be a time-consuming process.

1.4 Contributions

This dissertation explores the conjecture that many Internet attacks entail network behavior that manifests quite differently from that of a legitimate host. Accordingly, we hypothesize that stochastic models that capture a host's network behavior provide a good foundation for identifying malicious network activity in real-time.

The overall structure of the dissertation is as follows:

- We investigate three malicious network activities: portscan, scanning worm infection, and targeting worm propagation. For each case, we examine real network trace data that contain both large samples of benign traffic and some instances of malicious traffic of interest if available.

- We develop detection algorithms based on the mathematical framework of sequential analysis founded by Wald in his seminal work [80]. Using a connection between the real-time detection of malicious network activity and a theory of sequential analysis (or sequential hypothesis testing), we show that one can often model a host's network activity as one of two stochastic processes, corresponding respectively to the access patterns of benign hosts and malicious ones.

  The detection problem then becomes one of observing a particular trajectory and inferring from it the most likely classification for the given host's behavior. This

approach enables us to estimate an algorithm's performance based on the measurable statistics of a host's traffic. More importantly, it provides an analytic framework, with which one can balance the goals of promptness and accuracy in detecting malicious network activity.

- We evaluate each algorithm's performance in terms of accuracy and detection speed using network trace data collected from multiple vantage points. While our algorithms require only connection-level information (source and destination IP addresses/ports, timestamp, and connection status), we assess the correctness of flagged cases using various methods such as payload inspection, hostname, and site-specific information.

Our contributions in this dissertation are:

1. **Portscan Detection:** We present an algorithm that rapidly detects remote hosts performing a portscan to a target network. Our algorithm, **Threshold Random Walk (TRW)**, is motivated by the empirically observed disparity between the frequency with which connections to newly visited local addresses are successful for benign hosts vs. for portscanners. To our knowledge, TRW is the first algorithm that provides a quantitative framework explaining the detection speed and the accuracy of port scan detection. Using trace-driven simulations, we show that TRW is more accurate and requires a much smaller number of events (4 or 5 in practice) to detect malicious activity compared to previous schemes. We published TRW and the performance results in [36].

2. **Detection of Scanning Worm Infections:** We present a hybrid approach that accurately detects scanning worm infection promptly after the infected local host begins to engage in worm propagation. Our approach integrates significant improvements that we have made to two existing techniques: sequential hypothesis testing and connection rate limiting. We show that these two extensions are vital for the fast containment of scanning worms:

   - **Reverse Sequential Hypothesis Testing:** To address a problem when an observed host can change its behavior, from benign to malicious, we evaluate the likelihood ratio of a given host being malicious in reverse chronological order of the observations. We prove that the reverse sequential hypothesis testing is equivalent to the forward sequential hypothesis testing with modifications in the likelihood ratio computation. This modified forward sequential hypothesis test significantly simplifies algorithmic implementation. Moreover, it has been shown to be optimal for change point detection [48].
   - **Credit-based Rate Limiting:** To ensure that worms cannot propagate rapidly between the moment scanning begins and the time at which a sufficient number of connection failures has been observed by the detector, we develop an adaptive scheme that limits the rate at which a host issues new connections that are indicative of scanning activity. Our trace-driven simulation results show

that the credit-based rate limiting causes much fewer unnecessary rate limitings than the previous scheme, which relies on a fixed 1 new connection per second threshold.

We published the scanning worm detection approach and the results in [57].

3. **Detection of Targeting Worm Propagations:** We present an algorithm that promptly detects local hosts engaged in targeting worm propagation. Our algorithm, **Rate-Based Sequential Hypothesis Testing (RBS)**, is built on the rate at which a host initiates a connection to a new destination. RBS promptly detects hosts that exhibit abnormal levels of fan-out rate, which distinguishes them from benign hosts as they are working through a list of potential victims. Derived from the theory of sequential hypothesis testing, RBS dynamically adapts its window size and threshold, based on which it computes a rate and updates its decision in real-time. We then present **RBS+TRW**, a unified framework for detecting fast-propagating worms independently of their target discovery strategy. This work is under submission.

Like misuse detection and specification-based detection, we need to build *a priori* models, but, unlike these two, we model not only malicious behavior but also benign behavior. In a broader sense, our approach is a form of statistical anomaly detection. However, while a typical anomaly detector raises an alarm as soon as the observation deviates what is modeled as normalcy, we continue observing traffic until it concludes one way or the other. In addition, to the degree we evaluated our algorithms, they work well without requiring training specific to the environment in which they are used.

We begin with a review of statistical hypothesis testing, which forms a basis of our detection algorithms. Then, we present a discussion of related work in Chapter 3, followed by three chapters presenting the algorithms and the results described above. Chapter 7 concludes the dissertation with implications for further efforts and directions for future research.

Chapter 2

Background

This dissertation concerns the problems of detecting a malicious host using statistical properties of the traffic that the host generates. In this chapter, we review hypothesis testing, a statistical inference theory for making rational decisions based on probabilistic information. We then discuss sequential hypothesis testing, which is the theory of solving hypothesis testing problems when the sample size is not fixed *a priori* and a decision should be made as the data arrive [8].

2.1 Hypothesis Testing

For simplicity, we consider a binary hypothesis testing problem, where there are two hypotheses, H_0 (benign) and H_1 (malicious). For each observation, the decision process must choose either hypothesis that best describes the observation.

Given two hypotheses, there are four possible outcomes when a decision is made. The decision is called a *detection* when the algorithm selects H_1 when H_1 is in fact true. On the other hand, if the algorithm chooses H_0 instead, it is called *false negative*. Likewise, when H_0 is in fact true, picking H_1 constitutes a *false positive*. Finally, picking H_0 when H_0 is in fact true is termed *nominal*.

The outcomes and corresponding probability notations are listed in Table 2.1. We use the detection probability, P_D, and the false positive probability, P_F, to specify performance conditions of the detection algorithm. In particular, for user-selected values α and β, we desire that:

$$P_F \leq \alpha \text{ and } P_D \geq \beta, \tag{2.1}$$

where typical values might be $\alpha = 0.01$ and $\beta = 0.99$.

Next, we need to derive a decision rule that optimizes a criterion by which we can assess the quality of the decision process. There are three well-established criteria: minimizing the average cost of an incorrect decision (Bayes' decision criterion); maximizing the probability of a correct decision; and maximizing the detection power constrained by the false positive probability (Neyman-Pearson criterion). A remarkable result is that all these three different criteria lead to the same method called the *likelihood ratio test* [35].

Table 2.1: Four possible outcomes of a decision-making process

Outcome	Probability notation	Description
Detection	P_D	$\Pr[\text{choose } H_1 \mid H_1 \text{ is true}]$
False negative	$1 - P_D$	$\Pr[\text{choose } H_0 \mid H_1 \text{ is true}]$
False positive	P_F	$\Pr[\text{choose } H_1 \mid H_0 \text{ is true}]$
Nominal	$1 - P_F$	$\Pr[\text{choose } H_0 \mid H_0 \text{ is true}]$

For a given observation, $\mathbf{Y} = (Y_1, \ldots, Y_n)$, we can compute the probability distribution function (or probability density function) conditional on two hypotheses. The likelihood ratio, $\Lambda(\mathbf{Y})$ is defined as:

$$\Lambda(\mathbf{Y}) \equiv \frac{\Pr[\mathbf{Y}|H_1]}{\Pr[\mathbf{Y}|H_0]} \tag{2.2}$$

Then, the likelihood ratio test compares $\Lambda(\mathbf{Y})$ against a threshold, η, and selects H_1 if $\Lambda(\mathbf{Y}) > \eta$, H_0 if $\Lambda(\mathbf{Y}) < \eta$, and either hypothesis if $\Lambda(\mathbf{Y}) = \eta$. The threshold value, η, depends on the problem and what criterion is used, but we can always find a solution [35].

2.2 Sequential Hypothesis Testing

In the previous section, we implicitly assume that the data collection is executed before the analysis and therefore the number of samples collected is fixed at the beginning of the decision-making process. However, in practice, we may want to make decisions as the data become available if waiting for additional observations incurs a cost. For example, for a real-time detection system, if data collection can be terminated after fewer cases, decisions taken earlier can block more attack traffic.

Wald formulated the theory of sequential hypothesis testing (or sequential analysis) in his seminal book [80], where he defined the sequential likelihood ratio test as follows: the likelihood ratio is compared to an *upper* threshold, η_1, and a *lower* threshold, η_0. If $\Lambda(\mathbf{Y}) \leq \eta_0$ then we accept hypothesis H_0. If $\Lambda(\mathbf{Y}) \geq \eta_1$ then we accept hypothesis H_1. If $\eta_0 < \Lambda(\mathbf{Y}) < \eta_1$ then we wait for the next observation and update $\Lambda(\mathbf{Y})$ (see Figure 2-1).

The thresholds η_1 and η_0 should be chosen such that the false alarm and detection probability conditions, (2.1) are satisfied. It is not *a priori* clear how one would pick these thresholds, but a key and desirable attribute of sequential hypothesis testing is that, for all practical cases, the thresholds can be set equal to simple expressions of α and β.

Wald showed that η_1 (η_0) can be upper (lower) bounded by simple expressions of P_F and P_D. He also showed that these expressions can be used as practical approximations for the thresholds, where the P_F and P_D are replaced with the user chosen α and β. Consider a sample path of observations $Y_1, \ldots, Y_n$, where on the n^{th} observation the upper threshold η_1 is hit and hypothesis H_1 is selected. Thus:

$$\frac{\Pr[Y_1, \ldots Y_n|H_1]}{\Pr[Y_1, \ldots Y_n|H_0]} \geq \eta_1$$

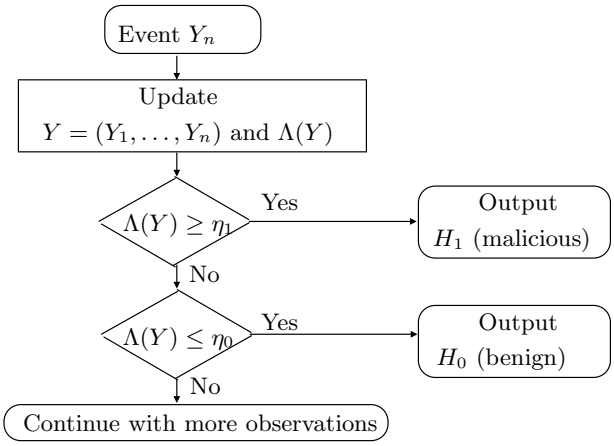


Figure 2-1: Sequential Likelihood Ratio Test

For any such sample path, the probability $\Pr[Y_1, \ldots Y_n | H_1]$ is at least η_1 times as big as $\Pr[Y_1, \ldots Y_n | H_0]$, and this is true for *all* sample paths where the test terminated with selection of H_1, regardless of *when* the test terminated (i.e., regardless of n). Thus, the probability measure of all sample paths where H_1 is selected when H_1 is true is at least η_1 times the probability measure of all sample paths where H_1 is selected when H_0 is true. The first of these probability measure (H_1 selected when H_1 true) is the detection probability, P_D, and the second, H_1 selected when H_0 true, is the false positive probability, P_F. Thus, we have an upper bound on threshold η_1:

$$\eta_1 \leq \frac{P_D}{P_F} \tag{2.3}$$

Analogous reasoning yields a lower bound for η_0:

$$\frac{1 - P_D}{1 - P_F} \leq \eta_0 \tag{2.4}$$

Now suppose the thresholds are chosen to be equal to these bounds, where the P_F and P_D are replaced respectively with the user-chosen α and β.

$$\eta_1 \leftarrow \frac{\beta}{\alpha} \qquad \eta_0 \leftarrow \frac{1 - \beta}{1 - \alpha} \tag{2.5}$$

Since we derived the bounds (2.3) and (2.4) for arbitrary values of the thresholds, these bounds of course apply for this particular choice. Thus:

$$\frac{\beta}{\alpha} \leq \frac{P_D}{P_F} \qquad \frac{1 - P_D}{1 - P_F} \leq \frac{1 - \beta}{1 - \alpha} \tag{2.6}$$

Taking the reciprocal in the first inequality in (2.6) and noting that since P_D is between

zero and one, $P_F < P_F/P_D$, yields the more interpretively convenient expression:

$$P_F < \frac{\alpha}{\beta} \equiv \frac{1}{\eta_1} \tag{2.7}$$

Likewise, for the second inequality in (2.6), noting that $1 - P_D < (1 - P_D)/(1 - P_F)$ yields:

$$1 - P_D < \frac{1-\beta}{1-\alpha} \equiv \eta_0 \tag{2.8}$$

Inequality (2.7) says that with the chosen thresholds (2.5), the actual false alarm probability, P_F, may be more than the chosen upper bound on false alarm probability, α, but not by much for cases of interest where the chosen lower bound on detection probability β is, say, 0.95 or 0.99. For example, if α is 0.01 and β is 0.99, then the actual false alarm probability will be no greater than 0.0101. Likewise, Inequality (2.8) says that one minus the actual detection probability (the miss probability) may be more than the chosen bound on miss probability, but again not by much, given that the chosen α is small, say 0.05 or 0.01. Finally, cross-multiplying in the two inequalities in (2.6) and adding yields:

$$1 - P_D + P_F \leq 1 - \beta + \alpha. \tag{2.9}$$

Inequality (2.9) suggests that although the actual false alarm or the actual miss probability may be greater than the desired bounds, they cannot *both* be, since their sum $1 - P_D + P_F$ is less than or equal to the sum of these bounds.

The above has taken the viewpoint that the user *a priori* chooses desired bounds on the false alarm and detection probabilities, α and β, and then uses the approximation (2.5) to determine the thresholds η_0 and η_1, with resulting inequalities (2.7) - (2.9). An alternative viewpoint is that the user directly chooses the thresholds, η_0 and η_1, with knowledge of the inequalities (2.7) and (2.8). In summary, setting the thresholds to the approximate values of (2.5) is simple, convenient, and within the realistic accuracy of the model.

Chapter 3

Related Work

The detection algorithms developed in this dissertation are concerned with two major malicious network activities: portscan activity and network virus and worm propagation. In this chapter, we first examine the traffic characteristics of these two malicious network activities. We show in later chapters that understanding the traffic patterns generated from these activities plays a key role in designing an effective detection algorithm. We then discuss previous defense methods and detection algorithms related to our work.

3.1 Portscan

Attackers routinely scan a target network to find local servers to compromise. Some network worms also scan a number of IP addresses in order to locate vulnerable servers to infect [44, 69]. Although portscanning itself may not be harmful, identifying a scanning source can facilitate several defense possibilities, such as halting potentially malicious traffic, redirecting it to other monitoring systems, or tracing the attackers.

There are two types of portscanning.[1] The first is *vertical* scanning where a portscanner probes a set of ports on the same machine to find out which services are running on the machine. The second is *horizontal* scanning where a portscanner probes multiple local addresses for the same port with the intention of profiling active hosts. In this study, we focus on detecting horizontal portscans, not vertical scans as the latter are easier to detect than the former, and require monitoring only a single host (herein, a portscan refers to a horizontal scan unless otherwise stated).

A single scan generates a short TCP connection or a short UDP flow. For TCP SYN scanning, a scanning host sends a TCP SYN (connection initiation) packet to a target host on a target port. If the target responds with a SYN ACK packet, the target is active and the corresponding port is open. If the target responds with RST, the target is up, but the port is closed. If there is no response received until timeout, the target host is unreachable or access to that service is blocked. For UDP scanning, a UDP response packet indicates that

[1] They can of course be combined and an attacker can probe a set of hosts using both vertical and horizontal scans.

the target port is open. When the target port is closed, a UDP scan packet elicits an ICMP unreachable message.

There are several other ways of scanning than TCP SYN scanning or UDP scanning. One example is a NULL scan [4] where a scanner sends out TCP packets with no flag bits set. If a target follows RFC 793 [53], it will respond with a RST packet only if the probe packet is sent to a closed port. Vivo *et al.* review TCP port scanners in detail including other stealthy scanning methods and coordinate scanning where multiple scanning sources are involved for probing [17]. This type of stealthy scanning can complicate implementing a portscan detector, but given that most stealthy scanning methods exploit unusual packet types or violate flow semantics, a detector that carefully watches for these corner cases can identify such a stealthy scanner.

When scanning a large network or all the possible open ports on a host (2^{16} possibilities), a scanning host that performs SYN scans or UDP scans generates a large number of short connections (or flows) destined to different IP addresses or ports. Additionally, to speed up a scanning process, those connections are separated by a small time interval, resulting in a high connection rate. In principle, these short connections to many different destinations (or ports) distinguish scanning traffic from legitimate remote access, and many detection systems use this pattern for identifying remote scanners [33, 56].

Another characteristic of scan traffic is that it tends to exhibit a greater number of failed connection attempts. Unlike benign users who usually access a remote host with a presumably legitimate IP address returned from a DNS server, scanners are opportunistic; failed access occurs when a chosen IP address is not associated with any active host, the host is not running a service of interest, or a firewall blocks incoming requests to the port that is being probed. This characteristic can be useful to detect scanners regardless of their scanning rate including a slow scanner that can evade a simple rate-based portscan detector such as Snort [56].

In Chapter 4, we examine these properties of scans using real trace data and develop a fast portscan detection algorithm that quickly identifies a remote port scanner generating disproportionally large number of failed connection attempts to many different local hosts.

3.1.1 Related Work on Scan Detection

Most scan detection is in the simple form of detecting N events within a time interval of T seconds. The first such algorithm in the literature was that used by the Network Security Monitor (NSM) [33], which had rules to detect any source IP address connecting to more than 15 distinct destination IP addresses within a given time window. Snort [56] implements similar methods. Version 2.0.2 uses two preprocessors. The first is packet-oriented, focusing on detecting malformed packets used for "stealth scanning" by tools such as *nmap* [4]. The second is connection-oriented. It checks whether a given source IP address touched more than X number of ports or Y number of IP addresses within Z seconds. Snort's parameters are tunable, but both fail in detecting conservative scanners whose scanning rate is sightly below than what both algorithms define as scanning.

Other work has built upon the observation that *failed* connection attempts are better indicators for identifying scans. Since scanners have little knowledge of network topology and system configuration, they are likely to often choose an IP address or port that is not active. The algorithm provided by Bro [52] treats connections differently depending on their service (application protocol). For connections using a service specified in a configurable list, Bro only performs bookkeeping if the connection attempt failed (was either unanswered, or elicited a TCP RST response). For others, it considers all connections, whether or not they failed. It then tallies the number of distinct destination addresses to which such connections (attempts) were made. If the number reaches a configurable parameter N, then Bro flags the source address as a scanner. Note that Bro's algorithm does not have any definite time window and the counts accumulate from when Bro started running.

By default, Bro sets $N = 100$ addresses and the set of services for which only failures are considered to HTTP, SSH, SMTP, IDENT, FTP data transfer (port 20), and Gopher (port 70). However, the sites from which our traces came used $N = 20$ instead.

Robertson *et al.* also focused on failed connection attempts, using a similar threshold method [55]. In general, choosing a good threshold is important: too low, and it can generate excessive false positives, while too high, and it will miss less aggressive scanners. Indeed, Robertson *et al.* showed that performance varies greatly based on parameter values.

To address problems with these simple counting methods, Leckie *et al.* proposed a probabilistic model to detect likely scan sources [40]. The model derives an access probability distribution for each local IP address, computed across all remote source IP addresses that access that destination. Thus, the model aims to estimate the degree to which access to a given local IP address is unusual. The model also considers the number of distinct local IP addresses that a given remote source has accessed so far. Then, the probability is compared with that of scanners, which are modeled as accessing each destination address with equal probability. If the probability of the source being an attacker is higher than that of the source being normal, then the source is reported as a scanner.[2]

A major flaw of this algorithm is its susceptibility to generating many false positives if the access probability distribution to the local IP addresses is highly skewed to a small set of popular servers. For example, a legitimate user who attempts to access a local personal machine (which is otherwise rarely accessed) could easily be flagged as scanner, since the probability that the local machine is accessed can be well below that derived from the uniform distribution used to model scanners.

In addition, the model lacks two important components. The first of these are confidence levels to assess whether the difference of the two probability estimates is large enough to safely choose one model over the other. Second, it is not clear how to soundly assign an *a priori* probability to destination addresses that have never been accessed. This can be particularly problematic for a sparsely populated network, where only small number of active hosts are accessed by benign hosts.

[2] In their scheme, no threshold is used for comparison. As long as the probability of a source being an attacker is greater than that of being normal, the source is flagged as a scanner.

The final work on scan detection of which we are aware is that of Staniford *et al.* on SPICE [66]. SPICE aims to detect stealthy scans—in particular, scans executed at very low rates and possibly spread across multiple source addresses. SPICE assigns anomaly scores to packets based on conditional probabilities derived from the source and destination addresses and ports. It collects packets over potentially long intervals (days or weeks) and then clusters them using simulated annealing to find correlations that are then reported as anomalous events. As such, SPICE requires significant run-time processing and is much more complex than our algorithm.

3.2 Network Virus and Worm Propagation

Fred Cohen first defined a computer "virus" as a program that can infect other programs by modifying them to include a possibly evolved copy of itself [16]. In 1983, Cohen demonstrated viral attacks with the modified `vd`, which he introduced to users via the system bulletin board. When a user executes the infected program, the infection process uses the user's permission to propagate to other parts of the Vax computer system. Surprisingly, the virus managed to grab all the system rights in under 30 minutes on average.

A network worm is a self-containing malware that automates an attack process by exploiting a common software vulnerability and propagates to other hosts *without* human intervention [63]. As a result, a carefully crafted network worm can spread over many vulnerable hosts at a high speed. However, some malicious programs use multiple propagation schemes combining the virus-like feature (relying on a user to trigger the infection) and the worm-like feature (self-replicating through vulnerable servers), thus blurring the line between virus and worm.

Factors that affect the propagation speed include break-in method, target discovery scheme, and payload propagation rate, as shown in Figure 3-1. In many cases, a worm spreads much faster than a virus as the former is capable of compromising a host almost immediately so long as the host is running a vulnerable program. However, locating such a vulnerable host requires an efficient searching method that will eventually lead a worm to reach most vulnerable servers on the Internet.

Table 3.1 lists notable computer viruses and worms that had widespread impact on the Internet in the past 20 years [21, 71, 87]. Many of them have variants that subsequently appeared shortly one after another, but for the sake of brevity, we discuss the most noteworthy ones.

As the first Internet-wide disruptive malware, the Morris worm affected about 5%-10% of the machines connected to the Internet back in 1988. Attacking various flaws in the common utilities of the UNIX system as well as easy-to-guess user passwords, the Morris worm effectively spread the infection to Sun 3 systems and VAX computers [20, 63]. The worm collects information on possible target hosts by reading public configuration files such as `/etc/hosts.equiv` and `.rhosts` and running a system utility that lists remote hosts that are connected to the current machine [63]

Melissa and Loveletter are both mass-mailing viruses. They propagate via an email attachment to the addresses found in an infected computer's address book. When a recipient

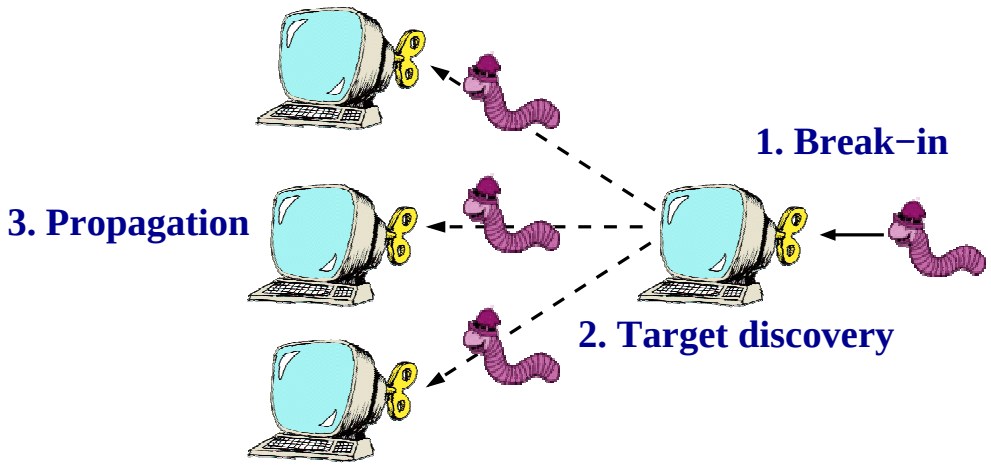


Figure 3-1: Virus and worm propagation

opens a viral attachment, the virus gets activated and compromises the user's machine. In addition to mass-mailing, the Loveletter virus overwrites local files with a copy of itself [23].

The Code Red worm infected more than 359,000 computers located all over the world on July 19, 2001 [45], exploiting a known buffer overflow vulnerability in Microsoft Internet Information Services (IIS) Web servers. Upon a successful infection, the worm picks a next victim at random from the IPv4 address space and attempts to send a malicious HTTP request that will trigger the buffer overflow bug. Because of its random target selection behavior, researchers were able to estimate the infected population by monitoring a large chunk of unused IP address space [45]. The worm is also programmed to launch a Denial-of-Service attack against the White House Web server.

Nimda uses multiple propagation vectors [27,72,12]. The first is that Nimda sends a crafted HTML email to the addresses harvested from local files. The malicious HTML email can be automatically executed if a user uses a vulnerable Microsoft Internet Explorer (IE) Web browser to view email. The second is that Nimda searches a vulnerable IIS Web server using random scanning and attempts to upload itself and to modify the server so that the server instructs a visitor to download the malicious executable. For 50% of the time, it picks a host in the same class B network and for the rest 50% of the time, it selects a host in the same class A network or a random host with the equal probability. This local preference scanning can increase a chance of finding an active server residing within the same administrative domain. The third is that Nimda embeds itself both to local and remote files, exchanging which will then spread the infection. Nimda is a hybrid malware that utilizes both the worm-like feature (i.e., actively spreading over other vulnerable hosts) and the virus-like feature (i.e., piggybacking on otherwise legitimate files to propagate).

In January 2003, the Slammer worm caused significant global slowdowns of the Internet with the massive amount of traffic from infected servers. The infinite loop in the worm code generates a random IP address and sends itself on UDP port 1434 to attack Microsoft SQL

server. The worm has a small payload (376 bytes) and its simple yet aggressive propagation method quickly infected most of the 75,000 victims in 10 minutes [44].

Several months later, the Blaster worm was unleashed attacking Microsoft Windows XP and 2000 operating systems that used an unpatched DCOM RPC service. The worm uses a sequential scanning with random starting points in order to search for vulnerable hosts. It is also programmed to launch a SYN flooding attack to the hard-coded URL, windowsupdate.com. However, the damage was not dramatic because Microsoft shifted the Web site to windowsupdate.microsoft.com [24].

After first being observed on January 26, 2004, the Mydoom virus quickly propagated over email and peer-to-peer file sharing network. Interesting features of this virus include a backdoor component and a scheduled Denial-of-Service attack on www.sco.com. The virus creates a backdoor on the infected machine and listens on the first available TCP port between 3127 and 3198. The backdoor enables a remote attacker to use the infected machine as a TCP proxy (e.g., spam relay) and to upload and execute arbitrary binaries [26, 85].

The Witty worm has the shortest vulnerability-to-exploit time window to date: the worm was unleashed within one day after the vulnerability was announced. It targets a machine running a vulnerable Internet Security Systems software. Witty carries a destructive payload that randomly erases disk blocks of an infected system [59].

In August 2005, the Zotob worm affected machines running a vulnerable Microsoft Windows Plug and Play service, which included many computers at popular media companies such as ABC, CNN and the New York Times. Zotob has a "bot" component that attempts to connect to Internet Relay Chat (IRC) channel at a pre-defined address, through which an attacker can remotely manipulate the infected machine. It also disables access to several Web sites of computer security companies that provide anti-virus programs [28, 88].

In summary, a network worm or virus exhibits a different network traffic behavior depending on the employed propagation method: Code Red and Slammer use a single TCP connection or a UDP packet to transmit an infection payload but generate a lot of scan traffic for target discovery. Nimda, Blaster and Zotob invoke multiple connections over different applications to spread the infection. However, this propagating behavior of a malware can have several conspicuous traffic patterns when compared to the network traffic generated from a typical benign application as the propagation is relatively rare in "normal" application use. We look into both target discovery methods and propagation schemes and develop suitable stochastic models capturing malicious network activity, which will be used as a basis of detecting malware propagation.

3.2.1 Related Work on Worm Detection

Moore *et al.* [46], model attempts at containing worms using quarantining. They perform various simulations, many of which use parameters principally from the Code Red II [13, 69] outbreak. They argue that it is impossible to prevent systems from being vulnerable to worms and that treatment cannot be performed fast enough to prevent worms from

spreading, leaving containment (quarantining) as the most viable way to prevent worm outbreaks from becoming epidemics.

Early work on containment includes Staniford *et al.*'s work on the GrIDS Intrusion Detection System [68], which advocates the detection of worms and viruses by tracing their paths through the departments of an organization. More recently, Staniford [65] has worked to generalize these concepts by extending models for the spread of infinite-speed, random scanning worms through homogeneous networks divided up into "cells". Simulating networks with 2^{17} hosts (two class B networks), Staniford limits the number of first-contact connections that a local host initiates to a given destination port to a threshold, T. While he claims that for most ports, a threshold of $T = 10$ is achievable in practice, HTTP and KaZaA are exceptions. In comparison, our reverse sequential hypothesis testing described in Chapter 5 reliably identifies HTTP scanning in as few as 10 observations.

Williamson first proposed limiting the rate of outgoing packets to new destinations [89] and implemented a virus throttle that confines a host to sending packets to no more than one new host a second [76]. While this slows traffic that could result from worm propagation below a certain rate, it remains open how to set the rate such that it permits benign traffic without impairing detection capability. For example, Web servers that employ content distribution services cause legitimate Web browsing to generate many concurrent connections to different destinations, which a limit of one new destination per second would significantly hinder. If the characteristics of benign traffic cannot be consistently recognized, a rate-based defense system will be either ignored or disabled by its users.

Numerous efforts have since aimed to improve the simple virus throttle by taking into account other metrics such as increasing numbers of ICMP host-unreachable packets or TCP RST packets [14], and the absence of preceding DNS lookups [84]. The TRAFEN [9, 10] system exploits failed connections for the purpose of identifying worms. The system is able to observe larger networks, without access to end-points, by inferring connection failures from ICMP messages. One problem with acting on information at this level is that an attacker could spoof source IP addresses to cause other hosts to be quarantined.

An approach quite similar to our own scanning worm detection algorithms has been simultaneously developed by Weaver, Staniford, and Paxson [83]. Their approach combines the rate limiting and approximated sequential hypothesis test with the assumption that connections fail until they are proved to succeed. While our modified forward sequential hypothesis testing is proved to be optimal [48], their scheme could cause a slight increase in detection delay, as the successes of connections sent before an infection event may be processed after the connections that are initiated after the infection event. In the context of their work, in which the high-performance required to monitor large networks is a key goal, the performance benefits are likely to outweigh the slight cost in detection speed.

There have been recent developments of worm detection using "content sifting" (finding common substrings in packets that are being sent in a many-to-many pattern) and automatic signature generation [37, 62]. Although constructing a right signature can be hard, it reduces chances of false alarms once a crisp signature is available. However, a signature-based detection has a limited ability to encrypted traffic when employed at a firewall. These

approaches are orthogonal to our approach based on traffic behavior in that the former require payload inspection, for which computationally intensive operations are often needed. However, our approach can supplement a signature-based detection by flagging a suspicious subset of traffic.

Table 3.1: Notable computer viruses and worms: V stands for virus and W stands for worm

Year	Name	Type	Exploits	Propagation Scheme
1988	Morris	W	Vulnerabilities in UNIX Sendmail, Finger, rsh/rexec; weak passwords	The worm harvests hostnames from local files and sends object files to a target machine. The target then opens a connection back to the originator, which creates a duplicate process in the target machine.
1999	Melissa	V	MS Word macro	The virus sends an email with a malicious attachment to the first 50 addresses found in the address book.
2000	LoveLetter	V	MS Visual Basic script	The virus sends an email with a malicious attachment to everyone in the address book.
2001	Code Red	W	MS IIS vulnerability	The worm sends a malicious HTTP payload to a randomly generated IP address on the TCP port 80.
	Nimda	W/V	MS IE and IIS vulnerabilities	Nimda sends itself by email or copies infected files to the open network shares and to vulnerable MS IIS Web servers via TFTP on the UDP port 69.
2003	Slammer	W	Buffer overflow bugs in MS SQL server and MSDE	The worm sends a malicious UDP packet to a randomly generated IP address on the port 1434.
	Blaster	W	RPC/DCOM vulnerability in Windows	The worm attempts to connect to a randomly generated IP address on the TCP port 135. Successful attack starts a shell on port 4444 through which the originator instructs the target. The target downloads the worm using the originator's TFTP server on the port 69
2004	Mydoom	V	Executable email attachment; peer-to-peer file sharing	The virus sends itself to harvested email addresses or copies itself to a KaZaA file sharing folder.
	Witty	W	Internet Security Systems software vulnerability	The worm sends a malicious UDP packet from the source port 40000 to randomly generated IP addresses.
2005	Zotob	W	MS Windows Plug & Play service vulnerability	The worm attempts to connect to a randomly generated IP address on the TCP port 445. Successful attack starts a shell on port 8888 through which the originator instructs the target. The target downloads the worm using the originator's FTP server on the port 33333.

Chapter 4

Portscan Detection

Many networks employ a network intrusion detection system (NIDS), which is usually placed where it can monitor all the incoming and outgoing traffic of the networks.[1] One of the basic functionalities that an NIDS provides is detecting a remote port scanner who tries to locate vulnerable hosts within the network. Figure 4-1 shows a typical scenario of portscan detection: a portscanner attempts to probe a /16 network whose IP network prefix is 18.31.0.0. A network intrusion detection system (NIDS) watches incoming and outgoing packets of the scanner and alerts the portscanning traffic at the N^{th} scan.

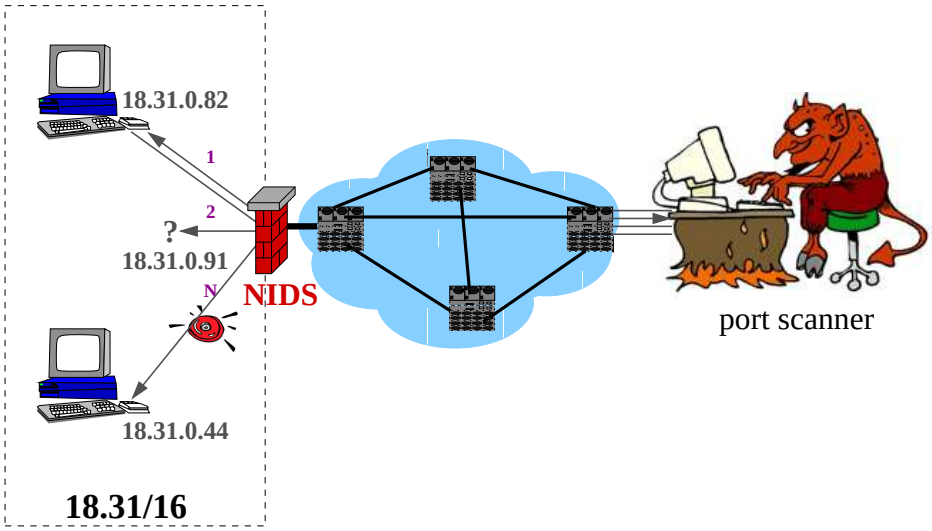


Figure 4-1: Network intrusion detection system

A number of difficulties arise, however, when we attempt to formulate an effective algorithm for detecting portscanning. The first is that there is no clear definition of the activity.

[1] Here we assume that these networks have well-defined and monitorable perimeters for installing an NIDS.

How to perform a portscan is entirely up to each scanner: a scanner can easily adjust the number of IP addresses to scan per second (scanning rate) and the number of IP addresses to scan per scanning host (scanning coverage). If we define portscanning as an activity of accessing more than 5 servers per second, we will miss any portscanners with a rate slower than 5 Hz.

There are also spatial and temporal considerations: do we want to aggregate activities originated from multiple IP addresses in order to detect "coordinated" scanning? Over how much time do we track activity? As time increases, a related spatial problem also arises: due to the use of DHCP, NAT, and proxies, a single address might correspond to multiple actual hosts, or, conversely, a single host's activity might be associated with multiple addresses over time.

Another issue is that of *intent*. Not all scans are necessarily hostile. For example, some search engines use not only "spidering" (following embedded links) but also portscanning in order to find Web servers to index. In addition, some applications (e.g., SSH, some peer-to-peer and Windows applications) have modes in which they scan in a benign attempt to gather information or locate servers. Ideally, we would like to separate such benign use from overtly malicious use. We note, however, that the question of whether scanning by search engines is benign will ultimately be a *policy* decision that will reflect a site's level of the desirability to have information about its servers publicly accessible.

The state of the art in detecting scanners is surprisingly limited. Existing schemes have difficulties catching all but high-rate scanners and often suffer from significant levels of false positives. In this work, we focus on the problem of *prompt* detection: how quickly after the initial onset of activity can we determine with high probability that a series of connections reflects hostile activity? Note that "quickly" here refers to the amount of subsequent activity by the scanner: the activity itself can occur at a very slow rate, but we still want to detect it before it has advanced very far, and ideally, do so with few false positives.

We begin with a formal definition of portscan activity based on the novel observation that the access pattern of portscanners often includes non-existent hosts or hosts that do not have the requested service running. Unlike the scanning rate or scanning coverage, this pattern of frequently accessing *inactive* servers is hard to alter since a portscanner has little knowledge of the configuration of a target network. On the other hand, this pattern rarely results from legitimate activity as a legitimate client would not send a packet to a destination unless there is reason to believe that the destination server accepts a request.

Regarding the spatial and temporal issues discussed above, for simplicity we confine our notion of "identity" to single remote IP addresses over a 24 hour period. The development of this chapter is as follows. In Section 4.1, we present the connection log data that motivate the general form of our detection algorithm. In Section 4.2, we develop the algorithm and present a mathematical analysis of how to parameterize its model in terms of expected false positives and false negatives, and how these trade off with the detection speed (number of connection attempts observed). We then evaluate the performance of the algorithm in Section 4.3, comparing it to that of other algorithms. We discuss directions for further work in Section 4.4 and summarize in Section 4.5.

Table 4.1: Summary of datasets

		LBL	ICSI
1	Data	Oct. 22, 2003	Oct. 16, 2003
2	Total inbound connections	15,614,500	161,122
3	Size of local address space	131,836	512
4	Active hosts	5,906	217
5	Total unique remote hosts	190,928	29,528
6	Scanners detected by Bro	122	7
7	HTTP worms	37	69
8	`other_bad`	74,383	15
9	*remainder*	116,386	29,437

4.1 Data Analysis

We grounded our exploration of the problem space, and subsequently the development of our detection algorithm, using a set of traces gathered from two sites, LBL (Lawrence Berkeley National Laboratory) and ICSI (International Computer Science Institute). Both are research laboratories with high-speed Internet connections and minimal firewalling (just a few incoming ports blocked). LBL has about 6,000 hosts and an address space of $2^{17} + 2^9 + 2^8$ addresses. As such, its host density is fairly sparse. ICSI has about 200 hosts and an address space of 2^9, so its host density is dense.

Both sites run the Bro NIDS. We were able to obtain a number of datasets of anonymized TCP connection summary logs generated by Bro. Each log entry lists a timestamp corresponding to when a connection (either inbound or outbound) was initiated, the duration of the connection, its ultimate state (which, for our purposes, was one of "successful," "rejected," or "unanswered"), the application protocol, the volume of data transferred in each direction, and the (anonymized) local and remote hosts participating in the connection. As the need arose, we were also able to ask the sites to examine their additional logs (and the identities of the anonymized hosts) in order to ascertain whether particular traffic did indeed reflect a scanner or a benign host.

Each dataset we analyzed covered a 24-hour period. We analyzed six datasets to develop our algorithm and then evaluated it on two additional datasets. Table 4.1 summarizes these last two; the other six had similar characteristics. About 4.4% and 42% of the address space is populated at LBL and ICSI respectively. Note that the number of active hosts is estimated from the connection status seen in the logs, rather than an absolute count reported by the site: we regard a local IP address as active if it ever generated a response (either a successful or rejected connection).

Among the 190,928 and 29,528 remote hosts that sent at least one packet to the corresponding site, the Bro system at the site flagged 122 (LBL) and 7 (ICSI) as scanners, using the algorithm described in the previous section with $N = 20$. Row 7 in Table 4.1 lists the number of remote hosts that were identified as attempting to spread either the "Code Red" or "Nimda" HTTP worm. Those remote hosts that happened to find a local Web server

and sent it the infection payload were caught by Bro based on the known signatures for the worms. However, it is important to note that the datasets may contain many more remote HTTP worms that were undiagnosed by Bro because in the course of their random scanning they did not happen to find a local HTTP server to try to infect.

The `other_bad` row in Table 4.1 corresponds to remote hosts that sent any packet to one of the following ports: 135/tcp, 139/tcp, 445/tcp, or 1433/tcp. These correspond to Windows RPC, NetBIOS, SMB, and *SQL-Snake* attacks (primarily worms, though Bro lacks detectors for non-HTTP worms), and they are blocked by the (very limited) firewalls at each site. It is important to note that the Bro monitor at LBL was located *outside* the firewall, and so would see this traffic; while that at ICSI monitored *inside* the firewall, so it did not see the traffic, other than a trickle that came from other nearby sites that were also within the firewall.

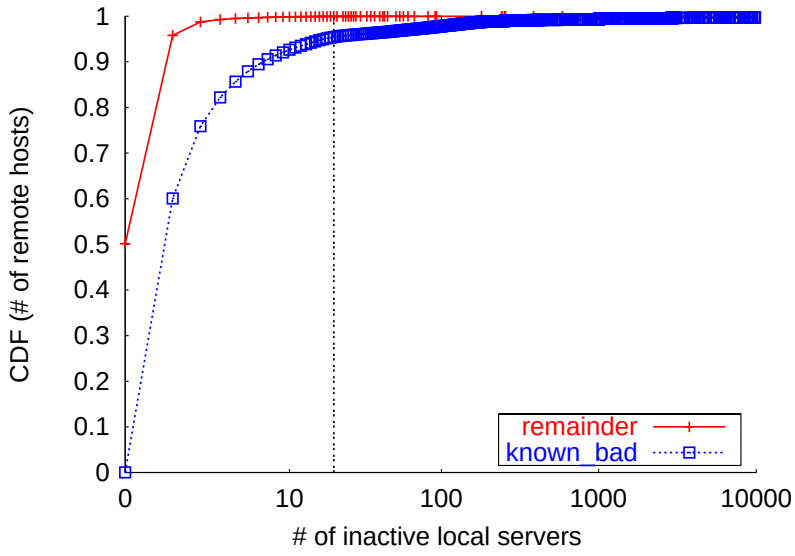
We will refer to the collection of the scanners, HTTP worms, and `other_bad` collectively as `known_bad`.

4.1.1 Separating Possible Scanners

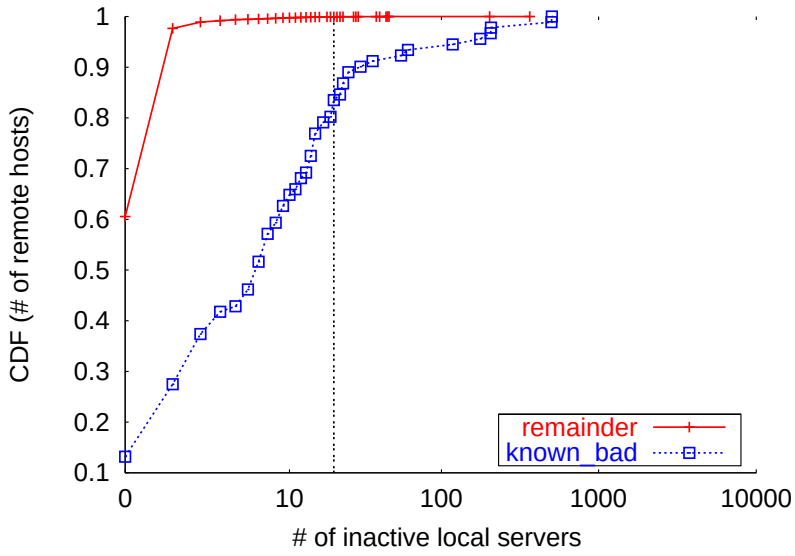
The available datasets give us a limited form of "ground truth," in that the remote hosts tagged as scanners very likely do reflect hostile scanners, and many (but surely not all) of the remote hosts tagged as benign are in fact benign. However, to soundly explore the data we need to have as strong a notion of ground truth as possible. In particular, we need some sort of determination as to which of the large number of *remainder* entries (row 9 of Table 4.1) are indeed undetected scanners that we then need to separate out from the set of otherwise-presumed-benign hosts before evaluating the effectiveness of any algorithm we develop.

This is a difficult but crucial problem. We need to find a way to bootstrap our assessment of which of the remainder are likely, but undetected (due to their lower level of activity), scanners. Ideally, the means by which we do so would be wholly separate from our subsequently developed detection algorithm, but we were unable to achieve this. Consequently, our argument is nearly circular: we show that there are properties we can plausibly use to distinguish likely scanners from non-scanners in the *remainder* hosts, and we then incorporate those as part of a (clearly imperfect) ground truth against which we test an algorithm we develop that detects the same distinguishing properties. The soundness of doing so rests in part in showing that **the likely scanners do indeed have characteristics in common with known malicious hosts**.

We first attempt to detect likely scanners by looking for remote hosts that make failed connection attempts to a disproportionate number of local addresses, comparing the distribution of the number of distinct inactive local servers accessed by `known_bad` hosts vs. those accessed by the as-yet undifferentiated *remainder*. Ideally, the distribution for *remainder* would exhibit a sharp modality for which one mode resembles `known_bad` hosts and the other is quite different. We could then use the mode as the data for distinguishing undiagnosed scanners from benign hosts, constructing a more accurate ground truth.



(a) LBL



(b) ICSI

Figure 4-2: Cumulative distribution of the number of remote hosts over the number of distinct local addresses to which a remote host unsuccessfully attempted to connect (# of inactive local servers). The vertical dotted line at $x = 20$ corresponds to the threshold used by the Bro NIDS at the sites.

Figure 4-2 plots this comparison. Unfortunately, we do not see the desired modality for *remainder*. Furthermore, the distribution of `known_bad` is such that in order to detect most of them solely on the data of their failed access attempts, we would need to use a threshold significantly lower than 20; and doing so will also flag a non-negligible portion[2] of *remainder* without us knowing whether this judgment is correct. Finally, we note that a basic reason for the large spread in the distributions in Figure 4-2 (note that the X-axis is log-scaled) is due to the very large spread we observe for the *rate* at which different scanners scan.

However, we *do* find a strong modality if we instead examine the ratio of hosts to which failed connections are made vs. those to which successful connections are made. For a given remote host, define ϕ_f as the percentage of the local hosts that the remote host has accessed for which the connection attempt failed (was rejected or unanswered).

$$\phi_f(\%) = 100 \times \frac{\text{\# of distinct local hosts where failed connection attempts are made}}{\text{\# of distinct local hosts where all connection attempts are made}}$$

Figure 4-3 shows the distribution of ϕ_f for each dataset. Figure 4-3(a) shows that about 99.5% of LBL's `known_bad` remotes hit nothing but inactive servers (expected, due to the firewall for most of the ports such hosts attempt to access). For ICSI, the proportion is spread between 60%–100%, but this still shows a clear tendency that `known_bad` hosts are likely to hit many non-existent hosts or hosts that do not support the requested service.

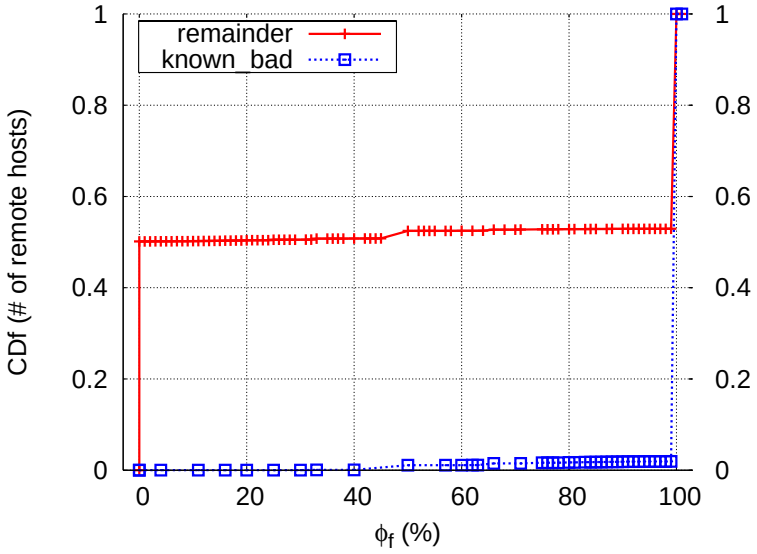
On the other hand, we see that in both cases, the *remainder* are sharply divided into two extreme sets—either 0% ϕ_f, or 100% ϕ_f—which then gives us plausible grounds to use this dichotomy to consider the latter remotes as likely to be scanners.

Based on this observation, we formulate the rule that *remainder* hosts with $< 80\%$ ϕ_f are potentially benign,[3] while hosts with higher values of ϕ_f will be treated as possible scanners. We term these latter as `suspect`. Table 4.2 summarizes the resulting classifications, and also the proportion due to remote hosts accessing HTTP, since those dominate the *remainder*.

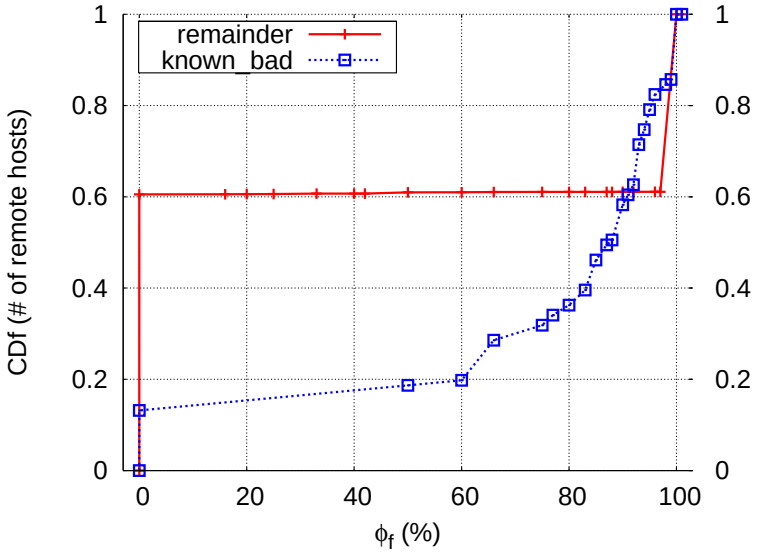
Interestingly, this simple criterion also allows us to catch the remote hosts that accessed many number of distinct local IP addresses. Figure 4-4 compares the distribution of the number of accessed IP addresses between `benign` and `suspect`. Note that there are few `benign` hosts that accessed more than 4 IP addresses. We conjecture them as web crawlers or proxies that often visits many web servers to collect link information or to provide a faster service to a large number of customers. Not to much our surprise, over 14% of `suspect` made connections more than 4 distinct IP addresses and some of them covered over 100 IP addresses in monitored networks.

[2]Recall that there are 10's of thousands of *remainder*, so even a small absolute portion of them can reflect a large number of hosts.

[3]Clearly, 80% is a somewhat arbitrary choice, but, given the sharp modality, any value greater than 50% has little effect on our subsequent results.

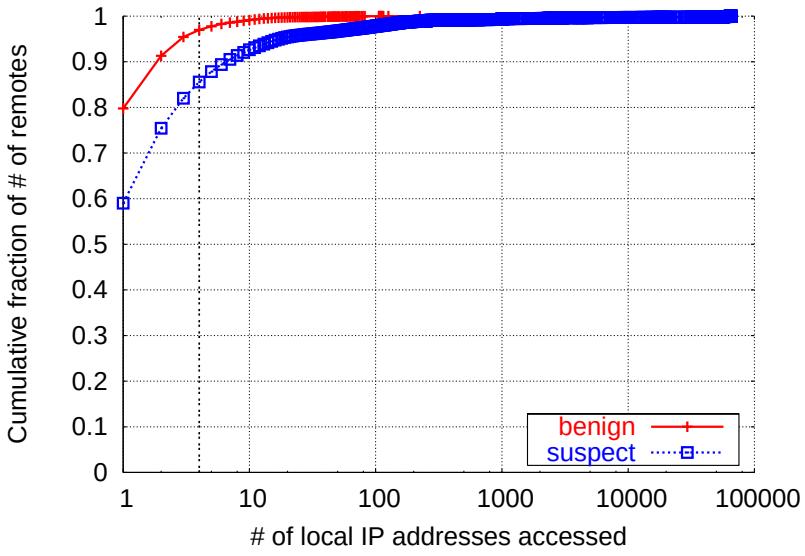


(a) LBL

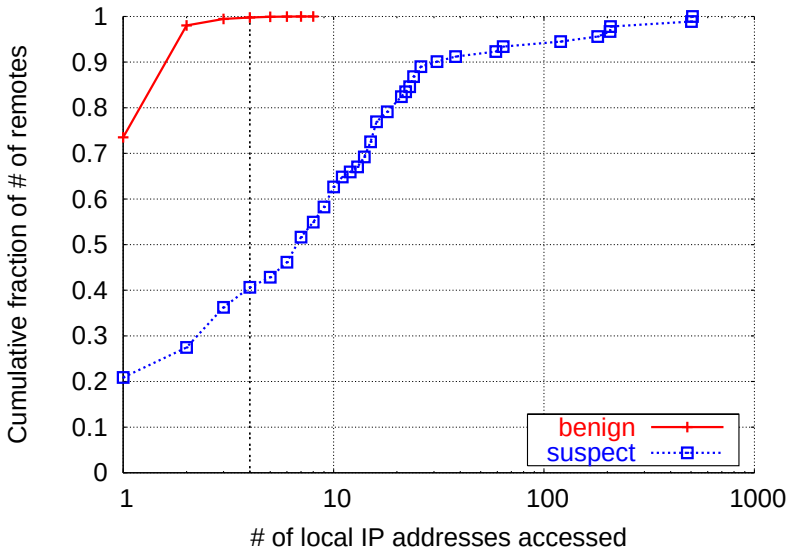


(b) ICSI

Figure 4-3: Cumulative distribution of the number of remote hosts over the percentage of the local hosts that a given remote host has accessed for which the connection attempt failed (ϕ_f)



(a) LBL



(b) ICSI

Figure 4-4: Cumulative distribution of the number of distinct IP addresses accessed per remote host: A vertical line corresponding $x = 4$ shows that very few benign hosts made connections to more than 4 distinct IP addresses.

Table 4.2: Remote host characteristics: $< 80\%$ ϕ_f is used to separate benign hosts from possible scanners. Here "scanners" are the remote hosts flagged by Bro as such.

			LBL	ICSI
Total unique remote hosts			190,928	29,528
`known_bad`			74,542	91
	scanners		122	7
	HTTP worms		37	69
	`other_bad`		74,383	15
remainder			116,386	29,437
	benign		61,465	17,974
		HTTP	47,343	6,026
	`suspect`		54,921	11,463
		HTTP	40,413	11,143

4.2 Threshold Random Walk: An Online Detection Algorithm

In the previous section, we showed that one of the main characteristics of scanners is that they are more likely than legitimate remote hosts to choose hosts that do *not* exist or do *not* have the requested service activated, since they lack precise knowledge of which hosts and ports on the target network are currently active. Based on this observation, we formulate a detection problem that provides the basis for an on-line algorithm whose goal is to reduce the number of observed connection attempts (compared to previous approaches) to flag malicious activity, while bounding the probabilities of missed detection and false detection.

4.2.1 Limitations of Simple Fixed-Size Windows

One simple way to detect a remote host exhibiting a high failure ratio is to gather N outcomes generated by the remote, to compute a failure ratio, and then to compare the ratio against a detection threshold. This fixed-size window approach, however, often results in suboptimal solutions: too small a window size may lead the algorithm to being susceptible to inaccurate decisions and too large a window size may unnecessarily slow down the detection.

Let us assume that we use 50% ϕ_f (defined in Section 4.1.1) as a threshold beyond which we trigger an alarm. Figure 4-5 illustrates two sample event sequences showing the first 10 outcomes in the order of their arrivals.

In the case of Figure 4-5(a), if we use a window size 5, then the first 5 outcomes will lead us to raise an alarm. But, if we have waited for 5 more outcomes, we would have reached a different decision. In general, decisions based on small number of observations are prone to errors especially when there are mixed signals observed within the window.

On the other hand, as shown in Figure 4-5(b), a large window can slow down the detection. In this example, we could have raised an alarm at the 5th outcome if we used a window size

(a) Window size: 5 (b) Window size: 10

Figure 4-5: Detections based on fixed-size windows: F represents a failure event and S a success event.

5. Basically, when the evidence observed so far strongly favors one model over the other, waiting to see more data does not increase a confidence level much and it is often better to terminate the observation now for faster detection.

Following sections show that our detection approach, which is built on the framework of the sequential hypothesis testing (see Chapter 2.2 for the review) overcomes the limitations of the simple fixed-size windows and dynamically adjusts a window size based on the strength of evidence observed.

4.2.2 Model

Let an event be generated when a remote source r makes a connection attempt to a local destination l. We classify the outcome of the attempt as either a "success" or a "failure", where the latter corresponds to a connection attempt to an inactive host or to an inactive service on an otherwise active host.

For a given r, let Y_i be a random (indicator) variable that represents the outcome of the first connection attempt by r to the i^{th} distinct local host, where

$$Y_i = \begin{cases} 0 & \text{if the connection attempt is a success} \\ 1 & \text{if the connection attempt is a failure} \end{cases}$$

As outcomes $Y_1, Y_2, \ldots$, are observed, we wish to determine whether r is a scanner. Intuitively, we would like to make this detection as quickly as possible, but with a high probability of being correct. Since we want to make our decision in real-time as we observe the outcomes, and since we have the opportunity to make a declaration after each outcome, the detection problem is well suited for the method of *sequential hypothesis testing* developed by Wald in his seminal work [80].

We consider two hypotheses, H_0 and H_1, where H_0 is the hypothesis that the given remote source r is benign and H_1 is the hypothesis that r is a scanner. Let us now assume that, conditional on the hypothesis H_j, the random variables $Y_i|H_j$ $i = 1, 2, \ldots$ are independent and identically distributed (i.i.d.). Then we can express the distribution of the Bernoulli

random variable Y_i as:

$$\begin{aligned} \Pr[Y_i = 0|H_0] = \theta_0, \quad \Pr[Y_i = 1|H_0] = 1 - \theta_0 \\ \Pr[Y_i = 0|H_1] = \theta_1, \quad \Pr[Y_i = 1|H_1] = 1 - \theta_1 \end{aligned} \tag{4.1}$$

The observation that a connection attempt is more likely to be a success from a benign source than a malicious one implies the condition:

$$\theta_0 > \theta_1.$$

The goal of the real-time detection algorithm is to make an early decision as an event stream arrives to the system while satisfying the performance conditions (2.1). As discussed in Chapter 2.2, upon each event's arrival, we calculate the likelihood ratio:

$$\Lambda(\mathbf{Y}) \equiv \frac{\Pr[\mathbf{Y}|H_1]}{\Pr[\mathbf{Y}|H_0]} = \Pi_{i=1}^{n} \frac{\Pr[Y_i|H_1]}{\Pr[Y_i|H_0]} \tag{4.2}$$

, where $\mathbf{Y}$ is the vector of events observed so far and $\Pr[\mathbf{Y}|H_i]$ represents the conditional probability mass function of the event stream $\mathbf{Y}$ given that model H_i is true; and where the second equality in (4.2) follows from the i.i.d. assumption. The decision-making process is illustrated in Figure 2-1.

Essentially, we are modeling a random walk for which the excursion probabilities come from one of two possible sets of probabilities, and we seek to identify which set is most likely responsible for an observed walk. We call our algorithm TRW, Threshold Random Walk, since our decision-making process corresponds to a random walk with two thresholds.[4]

As we will show in Section 4.2.4, while the Bernoulli distributions conditioned on hypotheses H_0 and H_1 play no role in the selection of the thresholds η_1 and η_0, they do (along with the thresholds) strongly affect N, the number of observations until the test terminates, i.e., until one of the hypotheses is selected.

4.2.3 Numerical Analysis

In order to assess the impact of approximate solutions for η_0 and η_1 (2.5) on the algorithm performance, we conducted numerical analysis varying parameters, α, β, θ_0, and θ_1. At the end of the simulation, we calculated achieved P_F and P_D and compared them with α and β to see whether the approximate solution for η_0 and η_1 allowed the algorithm to meet the performance criteria.

At each iteration, i, a random number, R_i, is generated from a uniform distribution $U(0, 1)$. Given α and β, two thresholds, η_0 and η_1 are calculated using Equation (2.5). R_i is used to determine whether this sample observation represents a *failed* event or *successful* event. If the simulation is to evaluate the detecting probability of scanners (benign users), R_i is

[4]To be more precise, it is a random walk in the logarithm of the likelihood ratio space.

compared to $\Pr[Y_i = 0|H_1] = \theta_1$ ($\Pr[Y_i = 0|H_0] = \theta_0$) and we regard the event as a success if R_i is less than a priori probability, θ_1 (θ_0).

Starting from 1, the likelihood ratio, Λ is updated as an event is generated. The decision is made at the point where Λ is greater than η_1 or less than η_0. This process is repeated until it accomplishes M decisions.

Figure 4-6 shows that regardless of *a priori* probabilities, θ_0 and θ_1, the performance goal is achieved when Equation (2.5) is used to set η_0 and η_1. It also shows that as θ_1 gets closer to θ_0, the performance goes down, but not lower than a target performance. This simulation result suggests the choice of *a priori* probabilities, θ_1 and θ_0 has little affect on the performance and the result still maintains the performance criteria. However, as we will show in the next section, *a priori* probabilities do affect the number of observations required to reach a decision and in practice, this can affect the performance since we can not always expect that such a large number of events are available for decision.

Figures 4-7(a) and 4-7(b) show the selected 100 steps during the simulations for detecting scanners or benign users. In both cases, $\beta = 0.99$, $\alpha = 0.01$, $\theta_0 = 0.9$, and $\theta_1 = 0.2$

For scanners, it requires at least three observations to reach a decision and the average number of observations is 4.110 when the simulation was done for 10,000 scanners. Out of 10,000 scanners, only 33 were detected as benign users, thus the resulting detection probability is 0.9967. Figure 4-7(a) shows the case for a false negative at around the 1760^{th} step where 4 consecutive successes happened for this particular scanner. The probability for this rare event is $0.2^4 = 0.0016$.

For benign users, it requires at least four observations to reach a decision and the average number of observations is 5.011 when the simulation was done for 10,000 benign cases. Out of 10,000 cases, 48 were decided as scanners, thus the resulting false positive probability is 0.0048. Figure 4-7(b) shows the case for a false positive at around the 860^{th} step where 3 failures within 4 trials happened for this particular benign user. The probability for this event is $0.1^3 \times 0.9 = 0.009$.

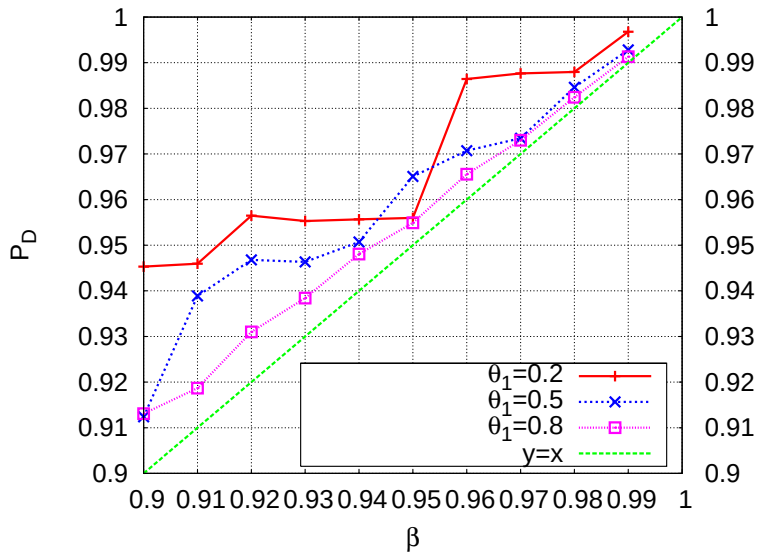
4.2.4 Number of Observations to Select Hypothesis

Given the performance criteria, (2.1), and the associated thresholds, (2.5), the remaining quantity of interest is the number of observation N until the test terminates, i.e., until one of the hypotheses is selected. Following Wald [80], we present approximate expressions for the expected value of N and the tail probability of N.

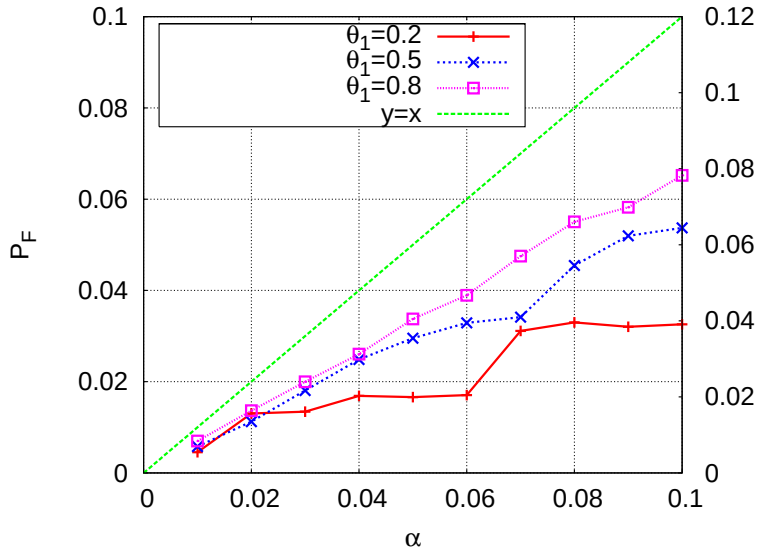
For the analysis of N, it is convenient to consider the log of the likelihood ratio, (4.2), and view the resulting expression as a random walk:

$$S_N \equiv \ln(\Lambda(\mathbf{Y})) = \sum_{i=1}^{N} X_i, \quad \text{where} \quad X_i \equiv \ln\left(\frac{\Pr[Y_i|H_1]}{\Pr[Y_i|H_0]}\right)$$

, and N is the observation number at which S_N first hits or crosses either the upper threshold, $\ln \eta_1$, or lower threshold, $\ln \eta_0$. (Note that $S_0 = 0$.)



(a) Actual detection rate (P_D) vs. target detection rate (β) : The target false positive rate (α) is set to 0.01 and θ_0 to 0.9.



(b) Actual false positive rate (P_F) vs. target false positive rate (α) : The target detection rate (β) is set to 0.99 and θ_0 to 0.9.

Figure 4-6: Achieved performance as a function of a target performance: Each dot in the plots is a simulation result (P_D or P_F) as we vary a target performance (β or α). The simulation result is obtained after $M = 100,000$ random numbers are generated. $y = x$ is a reference line indicating that a resulting performance is equal to a target performance.

From Wald's equality, $E[N] = E[S_N]/E[X_i]$, and we can obtain expressions for $E[S_N]$ and $E[X_i]$, conditioned on the hypotheses H_0 and H_1. Then, using the central limit theorem, we provide the tail probability of N, which can be useful to estimate the worst case scenarios when this algorithm is used.

For X_i,

$$X_i|H_0 = \begin{cases} \ln\frac{1-\theta_1}{1-\theta_0} & \text{with prob. } 1-\theta_0 \\ \ln\frac{\theta_1}{\theta_0} & \text{with prob. } \theta_0 \end{cases}$$

$$X_i|H_1 = \begin{cases} \ln\frac{1-\theta_1}{1-\theta_0} & \text{with prob. } 1-\theta_1 \\ \ln\frac{\theta_1}{\theta_0} & \text{with prob. } \theta_1 \end{cases}$$

$$\begin{aligned} E[X_i|H_0] &= (1-\theta_0)\ln\frac{1-\theta_1}{1-\theta_0} + \theta_0\ln\frac{\theta_1}{\theta_0} \\ E[X_i|H_1] &= (1-\theta_1)\ln\frac{1-\theta_1}{1-\theta_0} + \theta_1\ln\frac{\theta_1}{\theta_0} \end{aligned} \tag{4.3}$$

If we assume the sequential test ends with S_N hitting, equaling, either $\ln\eta_0$ or $\ln\eta_1$, i.e., if we ignore any overshoot, then

$$S_N|H_0 = \begin{cases} \ln\eta_1 & \text{with prob. } \alpha \\ \ln\eta_0 & \text{with prob. } 1-\alpha \end{cases}$$

$$S_N|H_1 = \begin{cases} \ln\eta_1 & \text{with prob. } \beta \\ \ln\eta_0 & \text{with prob. } 1-\beta \end{cases}$$

$$\begin{aligned} E[S_N|H_0] &= \alpha\ln\eta_1 + (1-\alpha)\ln\eta_0 \\ E[S_N|H_1] &= \beta\ln\eta_1 + (1-\beta)\ln\eta_0 \end{aligned} \tag{4.4}$$

Combining (2.5), (4.3), and (4.4), we obtain the approximate result in Equation (4.5).

$$\begin{aligned} E[N|H_0] &= \frac{\alpha\ln\frac{\beta}{\alpha} + (1-\alpha)\ln\frac{1-\beta}{1-\alpha}}{\theta_0\ln\frac{\theta_1}{\theta_0} + (1-\theta_0)\ln\frac{1-\theta_1}{1-\theta_0}} \\ E[N|H_1] &= \frac{\beta\ln\frac{\beta}{\alpha} + (1-\beta)\ln\frac{1-\beta}{1-\alpha}}{\theta_1\ln\frac{\theta_1}{\theta_0} + (1-\theta_1)\ln\frac{1-\theta_1}{1-\theta_0}} \end{aligned} \tag{4.5}$$

For the tail probability of N, we apply the central limit theorem to $\sum_{i=1} X_i$. Note that if the random walk, $\sum_{i=1}^{n_o} X_i$ is greater than or equal to upper threshold $\ln\eta_1$ at observation n_o, then the sequential hypothesis test must have terminated by then, i.e., $N \leq n_o$. Conditioning on the hypothesis for which hitting the upper threshold is more likely, H_1, we have:

$$\Pr[\sum_{i=1}^{n_o} X_i \geq \ln \eta_1 | H_1] \quad \leq \quad \Pr[N \leq n_o | H_1] \tag{4.6}$$

Normalizing the left hand side of (4.6) to mean zero variance one, yields:

$$\Pr\left[\frac{\sum_{i=1}^{n_o} X_i \ - \ n_o \mathrm{E}[X_i|H_1]}{\sqrt{n_o} \cdot \sigma(X_i|H_1)} \ \geq \ \frac{\ln \eta_1 \ - \ n_o \mathrm{E}[X_i|H_1]}{\sqrt{n_o} \cdot \sigma(X_i|H_1)} | H_1\right] \tag{4.7}$$

, where $\sigma(X_i|H_j)$ denotes the standard deviation of X_i given hypothesis H_j, $j = 0, 1$.

$$\begin{aligned} \sigma(X_i|H_0) &= \sqrt{\theta_0(1-\theta_0)} \cdot \ln\left(\frac{1-\theta_1}{1-\theta_0}\frac{\theta_0}{\theta_1}\right) \\ \sigma(X_i|H_1) &= \sqrt{\theta_1(1-\theta_1)} \cdot \ln\left(\frac{1-\theta_1}{1-\theta_0}\frac{\theta_0}{\theta_1}\right) \end{aligned}$$

Applying the central limit theorem to (4.7) yields an approximate lower bound for the distribution of $N|H_1$, which can be used as an approximation for the distribution itself, where the error tends to be on the conservative side (i.e., tends to under estimate the likelihood $N \leq n_o$). Thus,

$$\Pr[N \leq n_o | H_1] \ \approx \ 1 \ - \ \Phi\left(\frac{\ln \eta_1 \ - \ n_o \mathrm{E}[X_i|H_1]}{\sqrt{n_o} \cdot \sigma(X_i|H_1)}\right) \tag{4.8}$$

, where $\Phi(x)$ equals the probability of a normally distributed random variable with mean zero and variance one being less than or equal to x.

Analogous reasoning for the lower threshold and conditioning on H_0 yields

$$\Pr[N \leq n_o | H_0] \ \approx \ \Phi\left(\frac{\ln \eta_0 \ - \ n_o \mathrm{E}[X_i|H_0]}{\sqrt{n_o} \cdot \sigma(X_i|H_0)}\right) \tag{4.9}$$

4.2.5 Discussions on $E[N|H_1]$ vs. θ_0 and θ_1

As shown in Equation (4.5), $E[N|H_0]$ and $E[N|H_1]$ are a function of the four parameters, α, β, θ_0, and θ_1, the false positive and detection probabilities, and the degree to which scanners differ from benign hosts in terms of modeling their probability of making failed connections. With those values set, we can estimate the average number of distinct destination IP addresses that a given port scanner can probe before being caught by the algorithm.

Assuming a scanner picks IP addresses at random, θ_1—the probability that it chooses an IP address with the requested service on—depends on the density of these servers in a monitored network. Figure 4-8(a) shows how $E[N|H_1]$ changes as θ_1 increases. With $\alpha = 0.01$, $\beta = 0.99$, and $\theta_0 = 0.8$, $E[N|H_1]$ is 5.4 when $\theta_1 = 0.2$, and goes up to 11.8 when $\theta_1 = 0.4$ (we used $\theta_1 = 0.2$ based on the observations from data analysis in Section 4.1.) In general, it takes longer to tell one model from the other the closer the two

models are to each other. Figure 4-8(a) also shows that $E[N|H_1]$ goes up as α gets lower, which illustrates the trade off between low false positive probability and fast detection.

We can detect faster in situations where θ_0 is higher. Legitimate users often make a connection request with a host name. Unless the DNS provides outdated information, they rarely access inactive servers, and therefore θ_0—the probability that those users hit an active IP address—can be fairly high. However, the presence of benign Web crawlers and proxies that sometimes access inactive servers through broken links, or a few infected clients putting requests through a proxy that serves mostly benign users, can require a lower θ_0 for modeling.

In those circumstances where such problematic hosts can be controlled, however, then we can configure the detection algorithm to use a higher θ_0, and thus enable it to make a faster decision. Figure 4-8(b) shows $E[N|H_1]$ when θ_0 is set to 0.9. The decrease in detection time is significant.

In practice, one can tune θ_1 based on the density of the most popular service within an address space and then set θ_0 to a value reasonably higher than θ_1. However, for some networks where most addresses are allocated and running the same kind of service all the time, the little difference between θ_1 and θ_0 will render TRW less effective.

4.2.6 Limitations

We develop TRW based on the assumption that conditional on the hypothesis (that a remote host is benign or a scanner), any two distinct connection attempts will have the same likelihood of succeeding and their chances of success are unrelated to each other.

The bounds for upper and lower thresholds, Equations (2.3) and (2.4), are valid, given that the sequential hypothesis test will eventually terminate with probability one, which holds given independence of outcomes, and also for some cases of dependence [80]. Unfortunately, this will not hold for all cases of dependence. For instance, if a scanner probes N inactive servers exactly alternating with N active servers, our random walk will oscillate between one step up and one step down and it will never hit either threshold when $\theta_1 + \theta_0 = 1$. An informed attacker can evade TRW exploiting this dependence if he interleaves scan traffic with the traffic to N active hosts.

On the other hand, dependence that leads to positive correlation in outcomes (i.e., successes are more likely to be followed by another success or likewise for failures) will tend to shorten the time to hit a threshold. This form of dependence seems more likely to occur in practice.

Dependence, however, invalidates the second equality in Equation (4.2). Instead, the likelihood ratio should be calculated using a joint probability distribution, which complicates the computation.

4.3 Evaluation

This section evaluates the performance of the TRW algorithm in terms of its accuracy and the detection speed using trace-driven simulations. We explicate cases flagged as H_0

(benign) or H_1 (malicious) by TRW. Then, we compare the performance of TRW with that of Bro and Snort.

4.3.1 Trace-driven Simulation

We use the datasets described in Section 4.1 for evaluation. Each line in a dataset represents a connection seen by the Bro NIDS, sorted by the timestamp of the first packet belonging to the connection. Connection information includes a source IP, s, a destination IP, d, and the connection status. In reality, the connection status is not immediately available when the first packet arrives. For our analysis, we assume that the detector can consult an oracle that can tell upon seeing an incoming TCP SYN whether it will result in an established, rejected, or unanswered connection. (We might approximate such an oracle by giving the detector access to a database of which ports are open on which addresses, though "churn" at a site might make maintaining the database problematic.) Alternatively, the detector can wait a short period of time to see whether the SYN elicits a SYN ACK, a RST, or no response, corresponding to the three cases above.

For each s, TRW maintains 3 variables. D_s is the set of distinct IP addresses to which s has previously made connections. S_s reflects the decision state, one of: `PENDING`; H_0; or H_1. L_s is the likelihood ratio. For each line in the dataset, the simulation executes the following steps:

1) Skip the line if S_s is *not* `PENDING` (a decision has already made for the remote host s).

2) Determine whether the connection is successful or not. A connection is considered successful if it elicited a SYN ACK.[5]

3) Check whether d already belongs to D_s. If so, skip the next steps and proceed to the next connection. Update D_s with d, and update the likelihood ratio, L_s using Equation (4.2).

5) If L_s equals or exceeds η_1, set S_s to H_1. If L_s is lower than or equal to η_0, set S_s to H_0.

Table 4.3 shows the simulation results for LBL and ICSI datasets. The results depend on the parameter values; we present here results based on typical settings, where the detection probability (α) should be at least 0.99 and the false alarm rate (β) no larger than 0.01. We chose $\theta_1 = 0.2$ and $\theta_0 = 0.8$ based on the discussion in Section 4.1. Although we found

[5]Due to ambiguities in Bro's log format, for connections terminated by the remote originator with a RST we sometimes cannot determine whether the local host actually responded. Bro generates the same connection status for the case in which the connection was first established via the local host responding with a SYN ACK and the case where the remote host sent a SYN and then later, without receiving any reply, sent a RST. Accordingly, we treat such connections as failures if the logs indicate the local host did not send any data to the remote host.

Table 4.3: Simulation results when $\beta = 0.99$, $\alpha = 0.01$, $\theta_1 = 0.2$, and $\theta_0 = 0.8$

		LBL				ICSI			
Type		Count	P_D	$\overline{N}$	Max N	Count	P_D	$\overline{N}$	Max N
`scan`	Total	122	-	-	-	7	-	-	-
	H_1	122	1.000	4.0	6	7	1.000	4.3	6
`worm`	Total	32	-	-	-	51	-	-	-
	H_1	27	0.844	4.5	6	45	0.882	5.1	6
	`PENDING`	5	-	-	5	6	-	-	5
`other_bad`	Total	13,257	-	-	-	0	-	-	-
	H_1	13,059	0.985	4.0	10	0	-	-	-
	H_0	15	-	5.1	10	0	-	-	-
	`PENDING`	183	-	-	11	0	-	-	-
`benign`	Total	2,811	-	-	-	96	-	-	-
	H_1	33	-	8.1	24	0	-	-	-
	H_0	2,343	-	4.1	16	72	-	4.0	4
	`PENDING`	435	-	-	14	24	-	-	9
`suspect`	Total	692	-	-	-	236	-	-	-
	H_1	659	0.952	4.1	16	234	0.992	4.0	8
	`PENDING`	33	-	-	7	2	-	-	7

that almost all benign users never hit an inactive server, we chose θ_0 conservatively, to reduce the chances of flagging Web crawlers and proxies as scanners.

We excluded remote hosts that accessed less than 4 distinct local hosts from this table because with $\alpha = 0.99$, $\beta = 0.01$, $\theta_1 = 0.2$, and $\theta_0 = 0.8$, TRW requires at least 4 observations to make a decision. As a result, most of the remote hosts seen in the datasets (174,014 out of 190,928 for LBL and 29,138 out of 29,528 for ICSI) are not included in the table. These low profile remote hosts may contain very slow scanners that probe less than or equal to 3 IP addresses per day, and we note that a payload-based scheme may have a chance to detect these slow scanners if there is a signature available. However, many of those excluded remote hosts (114,313 out of 174,014 for LBL and 11,236 out of 29,138 for ICSI) failed to establish a single connection thus no payload information is available.

First, we group remote hosts into the categories defined in Section 4.1 and calculate P_D within each category. For both LBL and ICSI datasets, TRW caught all of the scanners flagged by Bro's algorithm. However, TRW missed a few HTTP worms that Bro identified (using known signatures), because of the slow scanning rate of those worms. Note that the maximum number of IP addresses scanned by those worms was 6 including at least 1 successful payload transmission for both the LBL and ICSI dataset.

TRW detected almost all the remote hosts that made connections to "forbidden" ports (see the corresponding rows for `other_bad`) and also the remote hosts classified as `suspect`. There were 15 `other_bad` flagged as H_0 for the LBL dataset. Among those 15 hosts, we observe that 11 remote hosts were machines that some local host had accessed at least once

before we flagged those remote hosts as H_0. These hosts are Microsoft Windows machines that sent NetBIOS packets back to a local host that initiated connections to them, which is a benign operation, and therefore it is correct to flag them as H_0. The other 3 were flagged as H_0 due to successful LDAP, IMAP4, or SMTP connections followed by a few NetBIOS packets. Although it hard to tell for sure whether these accesses reflect benign use or sophisticated multi-protocol probing, it is likely to be the former because the earlier connections succeeded.

This leaves just one more possible malicious remote host that missed being detected. Unfortunately, this one is difficult to distinguish because there were only 6 connections from that remote host recorded in the trace: 5 of them were very short, but successful, HTTP connections to 5 different servers, and there was only one unsuccessful connection attempt to port 135, which is generally perceived as hostile, but sometimes subject to "misfire". Many versions of Microsoft operating systems use port 135 for remote procedure calls. But, one of the vulnerabilities associated with this mechanism was exploited by the Blaster worm, which also propagates via port 135.

Surprisingly, there are no false positives for the ICSI dataset even though $\alpha = 0.01$. This is a rather encouraging result, demonstrating that TRW can outperform the performance specification in some cases. We selected a random subset of hosts belonging to `benign` and looked into their connections to confirm the result.

There are 33 false positives in the LBL dataset. On examination, we found that 3 of them sent out IDENT requests to a number of local machines in response to outbound SMTP or SSH connections. This is a common sequence of benign behavior. Since the IDENT requests were rejected by the local machines, the remote host was erroneously flagged as a scanner. This, however, can again be fixed if we keep track of remote hosts to which local hosts successfully established connections before the remote host makes failed connection attempts in response to those connections. We call these *friendly* hosts, and suggest using this additional context as a way to reduce false positives without changing any parameters of the general detection algorithm.

One host was an SMTP client that tried 4 different valid hosts in the monitored network, but terminated each connection with a RST packet 11 seconds after the initial SYN packet. From its hostname, it appears most likely a legitimate client, perhaps one working through a stale mailing list.

All of the remaining 29 false positives turned out to be Web crawlers and proxies. Dealing with these is problematic: crawlers are, after all, indeed scanning the site; and the proxies generally channel a mixture of legitimate and possibly malicious traffic. These might then call for a different reactive response from the NIDS upon detecting them: for example, using more stringent thresholds to require a larger proportion of scanning activity before they are shunned; or automatically releasing a block of the remote address after a period of time, in order to allow legitimate proxy traffic to again connect to the site.

Table 4.4 lists the types of `suspect` remote hosts that were flagged as H_1 by TRW. As discussed above, hosts flagged as H_1 due to responding IDENT connections instead are considered H_0. With the simple method suggested above of allowing remote hosts to make

Table 4.4: Break-down of "`suspects`" flagged as H_1

Type	LBL	ICSI
IDENT	18 (2.7%)	0 (0%)
$\geq$ 2 protocols	87 (13.2%)	8 (3.4%)
only HTTP	541 (82.1%)	226 (96.6%)
remainder	13 (2.0%)	0 (0%)

failed connections if they've previously received outbound connections from the site, we were able to confirm that all of the 18 `suspect` remote hosts were flagged due to responding IDENT connection for the LBL dataset. Over 80% made nothing but failed HTTP connections, and we therefore suspect them as undetected worms.

Table 4.3 also shows the average ($\overline{N}$) and maximum number of distinct local IP addresses that each detected remote host accessed upon being flagged. In theory, when $\alpha = 0.01$, $\beta = 0.99$, $\theta_0 = 0.8$, and θ_1 = 0.2, the approximate solution for $E[N|H_1]$ is 5.4 as shown in Section 4.2.5, and our trace-driven simulations are consistent with this figure. This suggests that the parameters chosen for θ_0 and θ_1 adequately model the actual behaviors of scanners and benign users. Note that with everything else fixed, $\overline{N}$ would have been much higher than 5 if θ_1 was greater than 0.3, as shown in Figure 4-8(a). It is also noteworthy that even in the worst case, a decision was made before a scanner probed more than 16 machines—strictly better than the best case provided by Bro's algorithm.

Table 4.5: Performance in terms of efficiency and effectiveness: *Post-filtering* eliminates remotes to which a local host previously connected. *Pre-filtering* is calculated based on Table 4.3.

		Trues	H_1	True positives	Efficiency	Effectiveness
LBL	Pre-filtering	14,103	13,900	13,867	0.998	0.983
	Post-filtering	14,068	13,878	13,848	0.998	0.984
ICSI	Pre-filtering	294	286	286	1.000	0.973
	Post-filtering	294	286	286	1.000	0.973

Finally, to quantify the effectiveness of TRW, we use the two measures proposed by Staniford *et al.* [66]:

- *Efficiency*: the ratio of the number of detected scanners (true positives) to all cases flagged as H_1.
- *Effectiveness*: the ratio of the number of true positives to all scanners (trues). This is the same as P_D, detection rate.

Efficiency conveys a similar meaning to false positive rate, but is more useful when the total number of true positives is significantly smaller than the total number of samples. Table 4.5

Table 4.6: Comparison of the number of H_1 across three categories for LBL dataset

		TRW			Bro			Snort		
Type	Total	H_1	$\overline{N}$	Max N	H_1	$\overline{N}$	Max N	H_1	$\overline{N}$	Max N
`scan`	121	121	4.0	6	121	21.4	28	63	16.8	369
`benign`	2811	30	-	-	0	-	-	57	-	-
`suspect`	692	659	4.1	16	0	-	-	28	7.9	33

Table 4.7: Comparison of the number of H_1 across three categories for ICSI dataset

		TRW			Bro			Snort		
Type	Total	H_1	$\overline{N}$	Max N	H_1	$\overline{N}$	Max N	H_1	$\overline{N}$	Max N
`scan`	7	7	4.3	6	7	35.9	119	5	6.0	6
`benign`	96	0	-	-	0	-	-	0	-	-
`suspect`	236	234	4.0	8	0	-	-	2	6.0	6

shows these values for the two sites. For ICSI, because of 8 misses (6 HTTP worms and 2 `suspect`), TRW results in a lower effectiveness (0.973) than expected ($\beta = 0.99$). But, the overall performance is excellent. We compare TRW's performance with that of Bro and Snort in the next section.

4.3.2 Comparison with Bro and Snort

For simplicity, we exclude the `worm` and `other_bad` category because as configured at LBL and ICSI, Bro does not perform scan-detection analysis for these. As throughout the chapter, we configure Bro's algorithm with $N = 20$ distinct hosts.

For Snort, we consider its `portscan2` scan-detection preprocessor, which takes into account distinct connections rather than distinct TCP SYN packets—the latter can generate many false positives if a single host sends multiple SYNs in the same failed connection attempt. We use Snort's default settings, for which it flags a source IP address that has sent connections to 5 different IP addresses within 60 seconds. (We ignore Snort's rule for 20-different-ports-within-60-seconds because our emphasis here is on detecting scans of multiple hosts rather than vertical scans of a single host.) We note that Snort's algorithm can erroneously flag Web crawlers or any automated process to fetch Web documents if there are more than 5 active Web servers in a monitored network. It can also be easily evaded by a scanner who probes a network no faster than 5 addresses/minute. Tables 4.6 and 4.7 show the number of (non-local) hosts reported as H_1 by the three algorithms.

Table 4.8 compares the efficiency and effectiveness across the three algorithms for both datasets. Note that two measures for TRW differ from Table 4.5 because of the two categories (`worm`, `other_bad`) excluded in this comparison. Bro has the highest efficiency followed by TRW and Snort. But Bro's highest efficiency comes at a cost of low effectiveness. Given its simple thresholds and limited time window, we expected that Snort would

Table 4.8: Comparison of the efficiency and effectiveness across TRW, Bro, and Snort

Trace	Measures	TRW	Bro	Snort
LBL	Efficiency	0.963	1.000	0.615
	Effectiveness	0.960	0.150	0.126
	$\overline{N}$	4.08	21.40	14.06
ICSI	Efficiency	1.000	1.000	1.000
	Effectiveness	0.992	0.029	0.029
	$\overline{N}$	4.06	36.91	6.00

provide fast detection. But, as shown in Tables 4.6 and 4.7, Snort was slower than TRW on average. In contrast to TRW, which on average flagged scanners when they hit no more than 5 distinct IP addresses, Snort waited for more than 13 IP addresses. Snort can increase the detection speed by lowering Y or Z values.[6] But, this will likely increase false alarms. Indeed, for LBL, 38.5% of the alarms by Snort were due to false positives.

Compared with Snort and Bro, TRW provided the highest effectiveness while maintaining higher than 0.96 efficiency. On average, detection was made when a target made connections to 4.1 active or inactive IP addresses. This average number of give-away IP addresses to scanners or `suspects` is about 3 times lower than that of Snort and about 5 times lower than that of Bro. In addition, TRW has the advantage over Snort that its analysis is not confined to a limited window of time: TRW has a wide dynamic range.

4.4 Discussion

In this section we look at a number of additional dimensions to the problem space and sketch our thinking on how each one can be pursued.

Leveraging Additional Information. TRW's performance is somewhat remarkable given the limited information it uses. Potential refinements include: (1) factoring in the specific service (for example, we could use more conservative parameters for possible HTTP scanning than for other ports, given the difficulty of confusing HTTP scanners with HTTP proxies); (2) distinguishing between unanswered connection attempts and rejected connection attempts, as the former might be more indicative of a complete "shot in the dark" whereas the latter could sometimes indicate a service that is temporarily off-line; (3) considering the time duration that a local address has been inactive, to be robust to benign connection attempts made to temporarily unavailable hosts; (4) considering the rate at which a remote host makes connection attempts (see Chapter 6); (5) introducing a component of correlation in the model, e.g., that two consecutive failed connection attempts are more suspect than two failures separated by a success; (6) devising a model of which local addresses and ports are historically more likely to be visited by benign sources or scanners.

[6] See Section 3.1.1 for the definitions of Y and Z: Snort raises an alarm if a given source IP address contacted more than X number of ports or Y number of IP addresses in Z seconds.

However, incorporating information such as the above is a two-edged sword. It may provide additional detection power—something to keep in mind for the discussion of other issues in this section—but at the cost of complicating use of the model, analysis of its properties, and, potentially, undermining its performance in some situations.

Managing State. The need to track for each remote host the different local addresses to which it has connected can in fact require a large amount of state. For example, imagine the operation of the algorithm during a SYN flooding attack with spoofed remote addresses. Virtually every arriving SYN will require the instantiation of state to track the new purported remote host. If, however, we cap the state available to the detector, then an attacker can launch a flood in order to exhaust the state, and then conduct a concurrent scan with impunity.

How to Respond. As shown in Section 4.3, TRW is much more effective at detecting low-volume scanners than Bro or Snort. However, this then raises the question of what to do with the alerts. For example, Table 4.5 shows that TRW detects nearly 14,000 scanners in the LBL dataset (presumably almost all of these are worms), vastly more than the 122 detected by Bro at the site. As mentioned in the Introduction, LBL uses Bro's scanner detection decisions to trigger *blocking* of the hostile remote host. However, the site reports that the blocking mechanism cannot scale to 1000's of blocks per day (this is why the site does not block HTTP scanners, because at times the endemic HTTP scans from worms can reach such levels). Thus, there is future work needed on mechanisms for determining whether a particular scanner is "block-worthy," i.e., will the given scanner continue to scan to a degree significant enough that they merit blocking or some form of rate control, or can they be ignored because they are scanning at a rate (or for a service of sufficiently low interest) that the site can afford to let the scan run its course?

Evasion and Gaming. Any scan detection algorithm based on observing failed connection attempts is susceptible to manipulation by attackers who spoof remote addresses and cause innocent remote hosts to be penalized. Depending on the reactive response taken when a scan is detected, address spoofing could provide the attacker with a great deal of leverage for denial-of-service. We note that the operators at LBL recognize this risk, and address it using "white lists" of critical remote hosts that should never be blocked. They have found this approach practical in today's environment, but this could change in the future if attackers become more energetic in targeting the response system. A possible additional approach here would be to have a honeypot respond to some of the connection attempts to see whether the remote host then completes the 3-way establishment handshake. If not, then the remote address is potentially spoofed.

Another issue concerns ways for an attacker to *evade* detection. For TRW, this is not so difficult. An attacker could compile a list of known servers at a site (running services other than those of interest to them) and then intermingle connection attempts to those with the wider connection attempts of a true scan. The successes of the camouflage connections would then drive the random walk away from an H_1 decision. Countering this threat requires either incorporating service information (as discussed above) or modeling which combinations of addresses legitimate users tend to access, and then giving less weight to successful connections not fitting with these patterns.

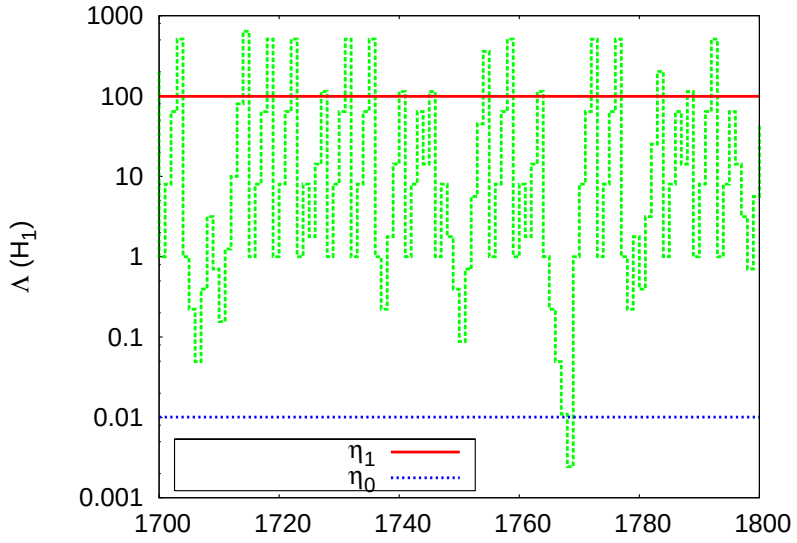
Distributed Scans. As stated in the Introduction, we confined our work to the problem of determining whether a single remote address corresponds to a malicious scanner. It appears difficult to directly adapt our framework to determining whether a set of remote addresses collectively correspond to malicious scanning (such as if they divide up the address space and each probe just a couple of addresses within it), because our algorithm depends on tracking success/failure information of individual remotes. It may, however, be possible to extend our algorithm with post processing to try to do so by combining a number of "low grade" signals (either detected scanners, or those whose random walks have taken them somewhat in the direction of H_1).

4.5 Summary

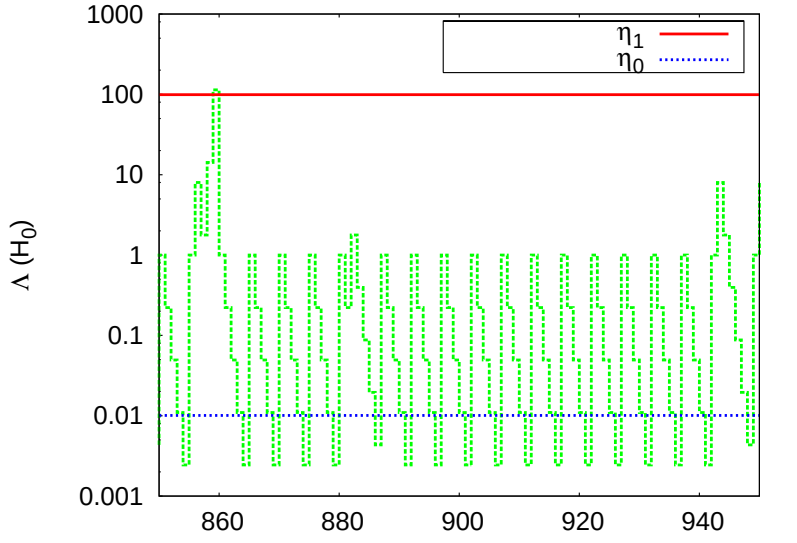
We have presented the development and evaluation of TRW—*Threshold Random Walk*—an algorithm to rapidly detect portscanners based on observations of whether a given remote host connects successfully or unsuccessfully to newly-visited local addresses. TRW is motivated by the empirically-observed disparity between the frequency with which such connections are successful for benign hosts vs. for known-to-be malicious hosts. The underpinnings of TRW derive from the theory of *sequential hypothesis testing*, which allows us to establish mathematical bounds on the expected performance of the algorithm.

Using an analysis of traces from two qualitatively different sites, we show that TRW requires a much smaller number of connection attempts (4 or 5 in practice) to detect malicious activity compared to previous schemes used by the Snort and Bro NIDS. TRW has the additional properties that (1) even though it makes quick decisions, it is highly *accurate*, with very few false positives, and (2) it is conceptually *simple*, which leads to both comprehensibility regarding how it works, and analytic tractability in deriving theoretical bounds on its performance.

In summary, TRW performs significantly faster and also more accurately than other current solutions.

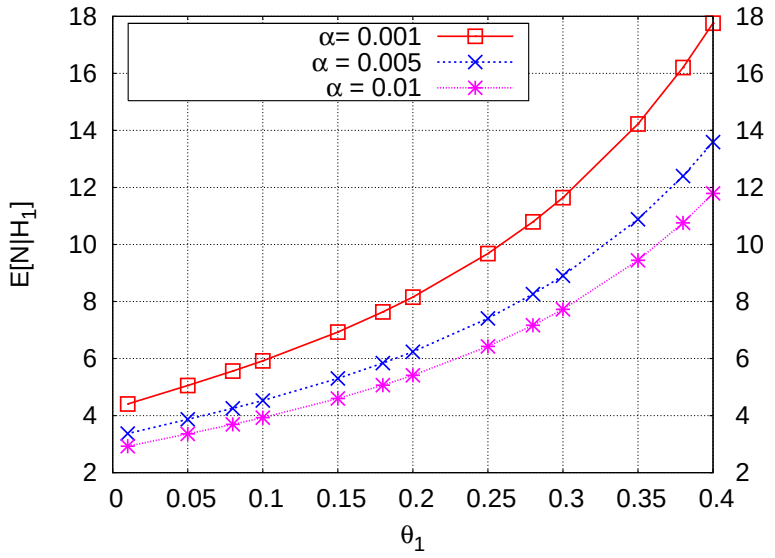


(a) Scanners

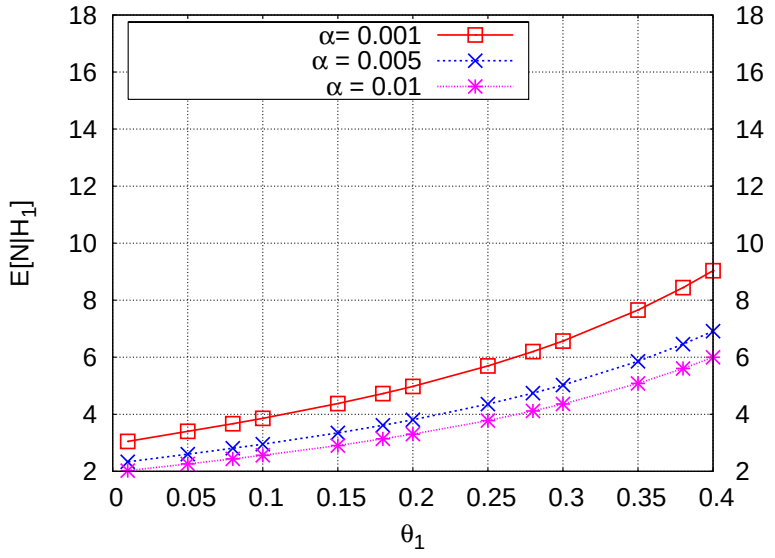


(b) Benign users

Figure 4-7: Simulation results: The graphs show how the likelihood ratio changes as random events are generated according to *a priori* probabilities, $\theta_0 = 0.9$ and $\theta_1 = 0.2$.



(a) $\theta_0 = 0.8$



(b) $\theta_0 = 0.9$

Figure 4-8: $E[N|H_1]$ vs. other parameters; β is fixed to 0.99

Chapter 5

Detection of Scanning Worm Infections

Network worms propagate to other hosts on the Internet via many methods: by searching the IPv4 address space (scanning worms); by searching the file system of an infected host for addresses that the infected host has already contacted (topological worms); by using the hist-lists of susceptible hosts identified by earlier scans (flash worms); or by querying specific servers for the addresses of potential victim hosts (meta-server worms). See [82] by Weaver *et al.* for more discussions of a taxonomy of network worms.

Among these propagation methods, scanning has been most frequently employed by the worms that have been released in the past few years. At its simplest, a scanning worm chooses its next victim by randomly drawing from the 2^{32} IPv4 addresses and transmits a malicious payload if the victim accepts a connection request [44, 45, 59]. A few variations were adapted by later worms in order to speed up the infection spread: Code Red II [11] and Sasser [73] probe local hosts that are in the same subnet as the infected machine more often than a completely random IP address. The Blaster worm [24] sequentially probes hosts starting from an IP address picked at random, with a bias toward local addresses on the same subnet.

The efficiency of scanning depends on the density of targeted vulnerable hosts on the Internet, which can be quite low for several reasons. First of all, more than 38% of IPv4 address blocks are not allocated to any registries [29], and therefore no hosts are reachable via those IP addresses. Second, many networks employ a firewall, which blocks incoming packets to certain ports that are known to be vulnerable to many existing exploits (e.g., TCP ports 135 and 139 corresponding to Windows RPC and NetBIOS). Finally, when a worm is fortunate enough to find an active host, the chosen host may not be running the targeted application.

To increase the chances of locating a vulnerable host, some scanning worms employ aggressive probing. For example, the Sasser worm creates 128 scanning threads to accelerate the searching process. The Slammer worm sends probing packets as fast as its bandwidth permits. The results are devastating: the Slammer worm rapidly spread to most of its victims in a matter of minutes, and its scanning traffic flooded many networks and shut down Internet services for hours [86].

For defense against these fast scanning worms, current proposals focus on *automated* responses to worms, such as quarantining infected machines [46], automatic generation and installation of patches [60, 61], and reducing the rate at which worms can issue connection requests so that a more carefully constructed response can be crafted [76, 89]. Yet even an automated response will be of little use if it fails to trigger soon after a host is infected. Infected hosts with high-bandwidth network connections can initiate thousands of connection requests per second, each of which has the potential to transmit the infection. On the other hand, an automated response that triggers too easily will erroneously identify hosts as infected, interfering with these hosts' reliable performance and causing significant damage.

We develop an approach that accurately detects scanning worm infection promptly after the infected host begins to engage in worm propagation. At its heart, the detection method uses the same intuition as the Threshold Random Walk (TRW) portscan detection algorithm discussed in Chapter 4. The intuition is that scanning worms are more likely than benign hosts to generate traffic to hosts that do not exist or do not have the requested service activated. TRW, which is designed to detect *inbound* scans initiated by hosts outside the local network, automatically adjusts the number of events to be collected with the strength of the evidence supporting the hypothesis that the observed host is, in fact, scanning.

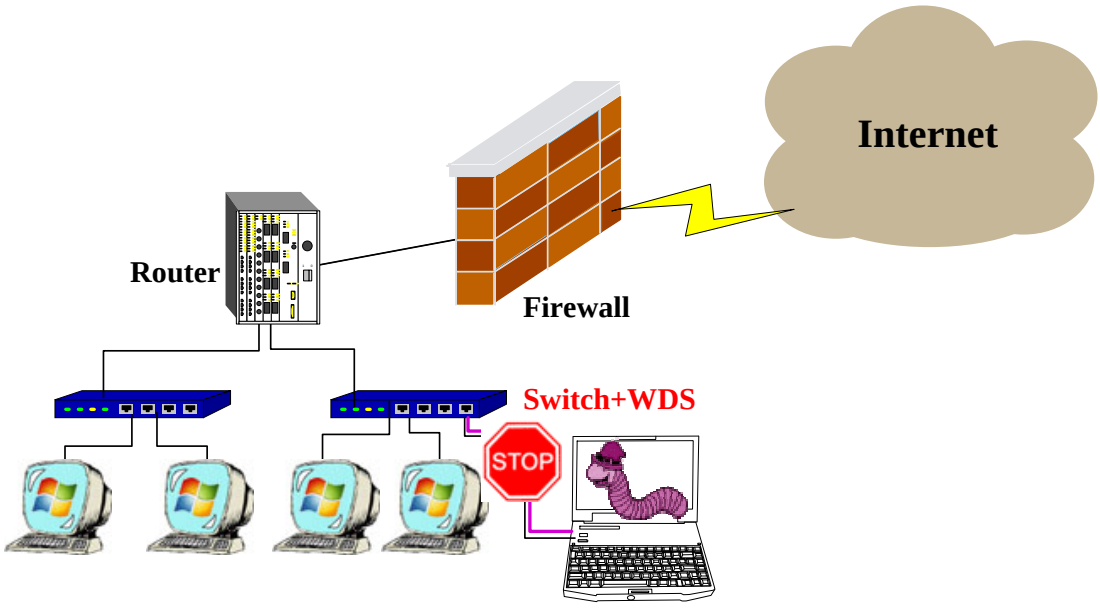


Figure 5-1: A Worm Detection System (WDS) is located to monitor a local network.

While TRW shows promise for quickly detecting scanning by hosts inside a local network, there are two significant hurdles to overcome. The first is that to determine whether a request to connect to a remote host will fail, one must often wait to see whether a successful connection response will be returned. Until a sufficient number of connection requests can be established as failures, a detector will lack the observations required to conclude that the system is infected. By the time the decision to contain the host is made, a worm with a high scan rate may have already spread to thousands of other hosts.

The second hurdle is that an observed host can change its behavior at any moment. TRW uses a single sequential hypothesis test per host and does not re-evaluate benign hosts over time. Unlike an intrusion detection system observing remote hosts, a worm detector is likely to observe benign traffic originating from an infected host before it is infected. It is therefore necessary to adapt this method to *continuously* monitor hosts for indications of scanning.

Figure 5-1 illustrates a network layout where one can set up a Worm Detection System (WDS), which monitors traffic generated from local hosts and automatically contains infected hosts to prevent further infection spread to other subnets within the local network. To detect infected hosts, the WDS need only process a small fraction of network events: a subset of connection request observations that we call *first-contact connection* requests and the responses to these requests that complete the connections. A first-contact connection request is a packet (TCP or UDP) addressed to a host with which the sender has not recently communicated.[1] These events are monitored because scans are mostly composed of first-contact connection requests.

In this work, we focus on detecting local hosts infected by a *scanning* worm, which searches through the IP address space for new victims. Hence, our approach is not likely to detect *targeting* worms that utilize other information about potentially vulnerable hosts. Examples of targeting worms are topological worms (e.g., Morris), flash worms, and meta-server worms (e.g., Santy).

In Section 5.1, we introduce a scan detection algorithm that we call a reverse sequential hypothesis test ($\overleftarrow{HT}$), and show how it can reduce the number of first-contact connections that must be observed to detect scanning.[2] This last-in, first-out event processing ensures that the number of observations $\overleftarrow{HT}$ requires to detect hosts' scanning behavior is not affected by the presence of benign network activity that may be observed before scanning begins.

In Section 5.2, we present a credit-based algorithm for limiting the rate at which a host may issue the first-contact connections that are indicative of scanning activity. This credit-based connection rate limiting (CBCRL) algorithm results in significantly fewer false positives (unnecessary rate limiting) than existing approaches.

When combined, this two-pronged approach is effective because these two algorithms are complementary. Without credit-based connection rate limiting, a worm could rapidly issue thousands of connection requests before enough connection failures have been observed by $\overleftarrow{HT}$ so that it can report the worm's presence. Because $\overleftarrow{HT}$ processes connection success and failure events in the order that connection requests are issued, false alarms are less likely to occur than if we used an approach purely relying on credit-based connection rate

[1] This definition of the first-contact connection request can be extended to differentiate servers running on a same host by taking a destination port into account. For example, two connection requests to a same destination host can be both first-contact connection requests by the extended definition if one goes to port 80 (Web server) and the other to port 25 (mail server).

[2] The letters in this abbreviation, $\overleftarrow{HT}$, stand for Hypothesis Testing and the arrow indicates the reverse sequential order in which observations are processed.

limiting, for which first-contact connections attempts are assumed to fail until the evidence proves otherwise.

We demonstrate the utility of these combined algorithms with the trace-driven simulations described in Section 5.3. The results of the simulations are presented in Section 5.4. The limitations of our approach, including strategies that worms could attempt to avoid detection, are presented in Section 5.5. We conclude with a discussion of future work in Section 5.6.

5.1 Reverse Sequential Hypothesis Testing

In this section, we present an on-line algorithm for detecting the presence of scanners within a local network by observing a host's network traffic. Here a scanner means a host sending out packets or connection initiation requests to a set of arbitrarily picked destination IP addresses on a given destination port. We use a sequential hypothesis test for its ability to adapt the number of observations required to make a decision to match the strength of the evidence it is presented with.

As with previous approaches to scan detection discussed in Chapter 4 and [55], we rely on the observation that only a small fraction of external addresses are likely to respond to a connection request at any given port. Benign hosts, which only contact systems when they have reason to believe that this connection request will be accepted, are more likely to receive a response to a connection request.

Recall that a first-contact connection request is a packet (TCP or UDP) addressed to a host with which the sender has not communicated for a given period of time. When a local host l initiates a first-contact connection request to a destination address, d, we classify the outcome as either a "success" or a "failure". If the request was a TCP SYN packet, the connection is said to succeed if a SYN-ACK is received from d before a timeout expires. If the request is a UDP packet, any UDP packet from d received before the timeout will do. We let Y_i be a random (indicator) variable that represents the outcome of the i^{th} first-contact connection request by l, where

$$Y_i = \begin{cases} 0 & \text{if the connection succeeds} \\ 1 & \text{if the connection fails} \end{cases}$$

As defined in Chapter 2.1, we call H_1 the hypothesis that host l is engaged in scanning (indicating infection by a worm) and H_0 the null hypothesis that the host is not scanning. Here l is a host in the local network, unlike in Chapter 4. We assume that, conditional on the hypothesis H_j, the random variables $Y_i|H_j$ $i = 1, 2, \ldots$ are independent and identically distributed. We can express the distribution of the Bernoulli random variable Y_i as:

$$\begin{aligned} \Pr[Y_i = 0|H_0] &= \theta_0, & \Pr[Y_i = 1|H_0] &= 1 - \theta_0 \\ \Pr[Y_i = 0|H_1] &= \theta_1, & \Pr[Y_i = 1|H_1] &= 1 - \theta_1 \end{aligned}$$

Given that connections originating at benign hosts are more likely to succeed than those initiated by a scanner, $\theta_0 > \theta_1$.

Sequential hypothesis testing chooses between two hypotheses by comparing the likelihoods that the model would generate the observed sequence of events, $\mathbf{Y}_n \equiv (Y_1, \ldots, Y_n)$, under each hypothesis. It does this by maintaining the ratio $\Lambda(\mathbf{Y}_n)$, the numerator of which is the likelihood that the model would generate the sequence of events $\mathbf{Y}_n$ under hypothesis H_1, and the denominator under hypothesis H_0.

$$\Lambda(\mathbf{Y}_n) \equiv \frac{\Pr[\mathbf{Y}_n|H_1]}{\Pr[\mathbf{Y}_n|H_0]} \tag{5.1}$$

We can write the change to $\Lambda(\mathbf{Y}_n)$ as a result of the i^{th}observation as $\phi(Y_i)$:

$$\phi(Y_i) \equiv \frac{\Pr[Y_i|H_1]}{\Pr[Y_i|H_0]} = \begin{cases} \frac{\theta_1}{\theta_0} & \text{if } Y_i = 0 \text{ (success)} \\ \\ \frac{1-\theta_1}{1-\theta_0} & \text{if } Y_i = 1 \text{ (failure)} \end{cases} \tag{5.2}$$

This formula and the i.i.d. assumption enable us to rewrite $\Lambda(\mathbf{Y}_n)$ inductively, such that $\Lambda(\mathbf{Y}_0) = 1$, and $\Lambda(\mathbf{Y}_n)$ may be calculated iteratively as each observation arrives.

$$\Lambda(\mathbf{Y}_n) = \prod_{i=1}^{n} \phi(Y_i) = \Lambda(\mathbf{Y}_{n-1})\phi(Y_n) \tag{5.3}$$

One compares the likelihood ratio $\Lambda(\mathbf{Y}_n)$ to an upper threshold, η_1, above which we accept hypothesis H_1, and a lower threshold, η_0, below which we accept hypothesis H_0. If $\eta_0 < \Lambda(\mathbf{Y}_n) < \eta_1$ then the result will remain inconclusive until more events in the sequence can be evaluated.[3] This is illustrated in Figure 5-2.

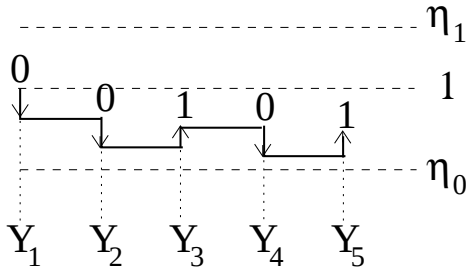


Figure 5-2: A log scale graph of $\Lambda(\mathbf{Y})$ as each observation, Y_i, is added to the sequence. Each success (0) observation decreases $\Lambda(\mathbf{Y})$, moving it closer to the benign conclusion threshold η_0, whereas each failure (1) observation increases $\Lambda(\mathbf{Y})$, moving it closer to the infection conclusion threshold η_1.

[3]Section 2.2 discusses how to set the thresholds in terms of a target false positive probability (α) and a target detection probability (β).

5.1.1 Detecting Infection Events

In TRW, we assumed that each *remote* host was either a scanner or benign for the duration of the observed period. When a host was determined to be benign it would no longer be observed. In contrast, in this work we are concerned with detecting infection events, in which a *local* host transitions from a benign state to an infected state. Should a host become infected while a hypothesis test is already running, the set of outcomes observed by the sequential hypothesis test may include those from both the benign and infected states, as shown in Figure 5-3. Even if we continue to observe the host and start a new hypothesis test each time a benign conclusion is reached, the test may take longer than necessary to conclude that an infection has occurred.

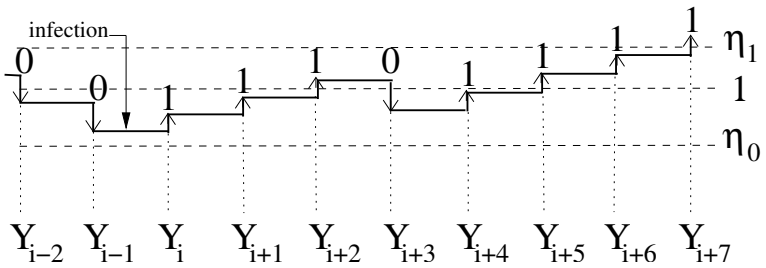


Figure 5-3: A log scale graph tracing the value of $\Lambda(\mathbf{Y})$ as it is updated for a series of observations that includes first-contact connection requests before (Y_{i-1} and Y_{i-2}) and after (Y_i and beyond) the host was infected.

The solution to this problem is to run a new sequential hypothesis test as each connection outcome is observed, evaluating these outcomes in reverse chronological order, as illustrated in Figure 5-4. To detect a host that was infected before it issued first-contact connection i (event Y_i), but after it had issued first-contact connection $i-1$, a reverse sequential hypothesis test ($\overleftarrow{HT}$) would require the same number of observations to detect the infection as would a forward sequential hypothesis that had started observing the sequence at observation i. Because the most recent observations are processed first, the reverse test will terminate before reaching the observations that were collected before infection.

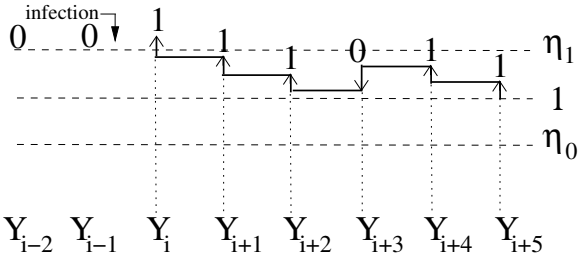


Figure 5-4: A log scale graph tracing the value of $\Lambda(Y_{i+5}, Y_{i+4}, \ldots)$, in which the observations in **Y** are processed in *reverse* sequential order. The most recent (rightmost) observation is the first one processed.

When we used the sequential hypothesis testing in Chapter 4 to detect scanning of a local network by remote hosts, the intrusion detection system could know *a priori* whether a connection would fail given its knowledge of the network topology and services. Thus, the outcome of a connection request from host i could immediately be classified as a success or failure observation (Y_i) and $\Lambda(\mathbf{Y}_n)$ could be evaluated without delay.

When a *local* host initiates first-contact connection requests to remote hosts, such as those shown in Figure 5-5, the worm detection system cannot immediately determine if the connection will succeed or fail. While some connection failures will result in a TCP RST packet or an ICMP packet [10], empirical evidence has shown that most do not [9]. The remaining connection attempts can be classified as failures only after a timeout expires.

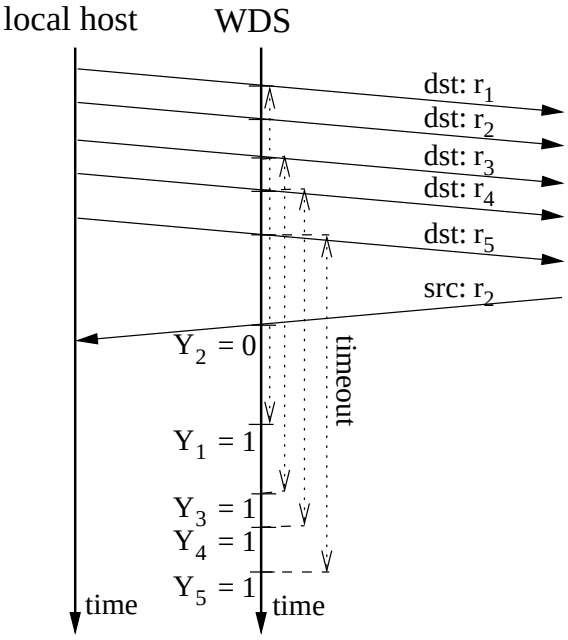


Figure 5-5: The success of first-contact connection requests by a local host to remote hosts cannot be established by the Worm Detection System (WDS) until a response is observed or a timeout expires.

While a sequential hypothesis test waits for unsuccessful connections to time out, a worm may send thousands of additional connection requests with which to infect other systems. To limit the number of outgoing first-contact connections, a sequential hypothesis testing approach must be paired with a method that limits connection initiation rates: the credit-based connection rate limiter described in Section 5.2 is one such scheme.

5.1.2 Optimizing the Computation of Repeated Reverse Sequential Hypothesis Tests

A straightforward implementation of the reverse sequential hypothesis testing requires that an arbitrarily large sequence of first-contact connection observations be stored. An iterative

computation of the likelihood ratio, Equation (5.3) does not apply for a reverse sequential hypothesis test. Instead, one has to maintain a sequence of observations from the beginning and step through a portion of this sequence each time a new observation is received in order to run a new test starting at that observation. The computational resources required for this implementation may be onerous.

In this section, we show that there exists an iterative function for reverse sequential hypothesis tests:

$$\bar{\Lambda}(\mathbf{Y}_n) = \max\left(1,\ \bar{\Lambda}(\mathbf{Y}_{n-1})\phi(Y_n)\right) \quad \bar{\Lambda}(\mathbf{Y}_0) \ \equiv \ 1$$

with state variable $\bar{\Lambda}(\mathbf{Y}_n)$, that can be calculated in the sequence in which events are observed, and that has the property that its value will exceed η_1 if and only if a reverse sequential hypothesis test would conclude from this sequence that the host was infected. ϕ is the likelihood ratio of an individual event as defined in Equation (5.2).

We first prove the following lemma stating that if a reverse sequential hypothesis test reports an infection, our optimized function will also report an infection.

Lemma 1. *For $\eta_1 > 1$ and for mutually independent random variables Y_i,*

$$\forall m \in [1, n] : \Lambda(Y_n, Y_{n-1}, \ldots, Y_m) \geq \eta_1 \Rightarrow \bar{\Lambda}(\mathbf{Y}_n) \geq \eta_1 \tag{5.4}$$

Proof. We begin by replacing Λ with its equivalent expression in terms of ϕ:

$$\eta_1 \leq \Lambda(Y_n, Y_{n-1}, \ldots, Y_m) = \prod_{i=m}^{n} \phi(Y_i) \tag{5.5}$$

We can place a lower bound on the value of $\bar{\Lambda}(Y_n)$ by exploiting the fact that, in any iteration, $\bar{\Lambda}$ cannot return a value less than 1.

$$\begin{aligned}
\bar{\Lambda}(\mathbf{Y}_n) &= \bar{\Lambda}(Y_1, Y_2, \ldots, Y_n) \\
&\geq 1 \cdot \bar{\Lambda}(Y_m, Y_{m+1}, \ldots, Y_n) \\
&\geq \prod_{i=m}^{n} \phi(Y_i) \\
&\geq \eta_1 \qquad \text{(using Equation (5.5))}
\end{aligned}$$

Thus, $\Lambda(Y_n, Y_{n-1}, \ldots, Y_m) \geq \eta_1 \Rightarrow \bar{\Lambda}(\mathbf{Y}_n) \geq \eta_1$. □

We must also prove that our optimized function will only report an infection when a reverse sequential hypothesis test would also report an infection. Recall that a reverse sequential hypothesis test will only report an infection if Λ exceeds η_1 before falling below η_0.

Lemma 2. *For thresholds $\eta_0 < 1 < \eta_1$ and for mutually independent random variables Y_i, if $\bar{\Lambda}(\mathbf{Y}_i) \geq \eta_1$ for some $i = n$, but $\bar{\Lambda}(\mathbf{Y}_i) < \eta_1$ for all $i \in [0, n-1]$, then there exists a subsequence of observations starting at observation n and moving backward to observation*

$m \in [0, n-1]$ *for which* $\Lambda(Y_n, Y_{n-1}, \ldots, Y_{m+1}) \geq \eta_1$ *and such that there exists no* k *in* $[m+1, n]$ *such that* $\Lambda(Y_n, Y_{n-1}, \ldots, Y_k) \leq \eta_0$.

Proof. Choose m as the largest observation index for which it held that:

$$\bar{\Lambda}(\mathbf{Y}_{m-1})\phi(Y_m) < 1$$

We know that $m < n$ because $\bar{\Lambda}(\mathbf{Y}_{n-1})\phi(Y_n)$ is greater than η_1 which is in turn greater than 1. Let $m = 0$ if the above relation does not hold for any observation with index greater than or equal to 1. It follows that $\bar{\Lambda}(\mathbf{Y}_m) = 1$ and thus:

$$\bar{\Lambda}(\mathbf{Y}_{m+1}) = \phi(Y_{m+1})$$

Because we chose m such that $\bar{\Lambda}(\mathbf{Y}_{j-1})\phi(Y_j) \geq 1$ for all $j > m$:

$$\begin{aligned} \bar{\Lambda}(\mathbf{Y}_n) &= \prod_{j=m+1}^{n} \phi(Y_j) \\ &= \Lambda(Y_n, Y_{n-1}, \ldots, Y_{m+1}) \end{aligned}$$

Thus, $\bar{\Lambda}(\mathbf{Y}_n) \geq \eta_1 \Rightarrow \Lambda(Y_n, Y_{n-1}, \ldots, Y_{m+1}) \geq \eta_1$.

To prove that there exists no k in $[m+1, n]$ such that $\Lambda(Y_n, Y_{n-1}, \ldots, Y_k) \leq \eta_0$, suppose that such a k exists. It follows that:

$$\prod_{j=k}^{n} \phi(Y_j) \leq \eta_0 < 1 \tag{5.6}$$

Recall that we chose m to ensure that:

$$\eta_1 \leq \prod_{j=m+1}^{n} \phi(Y_j) \tag{5.7}$$

The product on the right hand side can be separated into factors from before and after observation k.

$$\eta_1 \leq \prod_{j=m+1}^{k-1} \phi(Y_j) \cdot \prod_{j=k}^{n} \phi(Y_j) \cdot \tag{5.8}$$

We then use Equation (5.6) to substitute an upper bound of 1 on the latter product.

$$\eta_1 < \prod_{j=m+1}^{k-1} \phi(Y_j) \leq \bar{\Lambda}(\mathbf{Y}_{k-1})$$

This contradicts the hypothesis that $\bar{\Lambda}(\mathbf{Y}_i) < \eta_1$ for all $i \in [0, n-1]$. □

5.1.3 Algorithmic Implementation

Updating $\bar{\Lambda}$ for each observation requires only a single multiplication and two comparison operations. In fact, addition and subtraction operations are adequate as the iterative function is equivalent to $\Theta(\mathbf{Y}_n) = \max(0, \Theta(\mathbf{Y}_{n-1}) + \ln\phi(Y_n))$, where $\Theta(\mathbf{Y}_n) \equiv \ln\bar{\Lambda}(\mathbf{Y}_n)$. Because $\bar{\Lambda}$ is updated in sequence, observations can be discarded immediately after they are used to update the value of $\bar{\Lambda}$.

When running this algorithm in a worm detection system, we must maintain separate state information for each host being monitored. Thus, a state variable $\bar{\Lambda}_l$ is maintained for each local host l. It is also necessary to track which hosts have been previously contacted by l. We track the set of Previously Contacted Hosts, or PCH set, for each local host.

```
enum status {PENDING, SUCCESS, FAILURE};
struct FCC_Queue_Entry {
  ip4_addr DestAddr;
  time     WhenInitiated;
  status   Status;
}
```

Figure 5-6: The structure of entries in the First-Contact Connection (FCC) queue

Finally, each local host l has an associated queue of the first-contact connection attempts that l has issued but that have not yet been processed as observations. The structure of the records that are pushed on this FCC queue are shown in Figure 5-6. The choice of a queue for this data structure ensures that first-contact connection attempts are processed in the order in which they are issued, not in the order in which their status is determined.

The algorithm itself is quite simple and is triggered upon one of three events:

1. When the worm detection system observes a packet (TCP SYN or UDP) sent by local host l, it checks to see if the destination address d is in l's previously contacted host (PCH) set. If it isn't, it adds d to the PCH set and adds a new entry to the end of the FCC queue with d as the destination address and status `PENDING`. Note that the current algorithm does not act on a connection initiation packet if its destination address belongs to the PCH set in order to ensure that the algorithm processes only first-contact connections.

2. When an incoming packet arrives addressed to local host l and the source address is also the destination address (`DestAddr`) of a record in l's FCC queue, the packet is interpreted as a response to the first-contact connection request and the status of the FCC record is updated. The status of the FCC record is set to `SUCCESS` unless the packet is a TCP RST packet, which indicates a rejected connection.

3. Whenever the entry on the front of the FCC queue has status `PENDING` and has been in the queue longer than the connection timeout period, a timeout occurs and the entry is assigned the status of `FAILURE`.

When any of the above events causes the entry at the front of the FCC queue to have status other than `PENDING`, it is dequeued and $\bar{\Lambda}_l$ is updated and compared to η_1. If $\bar{\Lambda}_l \geq \eta_1$, we halt testing for host l and immediately conclude that l is infected. Dequeuing continues so long as $\bar{\Lambda}_l < \eta_1$, the front entry of the FCC queue has status other than `PENDING`, and the queue is not empty.

5.2 Credit-Based Connection Rate Limiting

It is necessary to limit the rate at which first-contact connections can be initiated in order to ensure that worms cannot propagate rapidly between the moment scanning begins and the time at which the scan's first-contact connections have timed out and been observed by our reverse sequential hypothesis test ($\overleftarrow{HT}$).

Twycross and Williamson [76, 89] use a technique they call a virus throttle to limit outgoing first-contact connections. When observing a given host, their algorithm maintains a working set of up to five hosts previously contacted by the host they are observing. In their work, a first-contact connection is a connection to a host that is not in this working set. First-contact connections issued when the working set is full are not sent out, but instead added to a queue. Once per second, the least recently used entry in the working set is removed and, if the pending queue of first-contact connection requests is not empty, a request is pulled off the queue, delivered, and its destination address is added to the working set. All requests in the queue with the same destination address are also removed from the queue and delivered.

Virus throttling is likely to interfere with HTTP connection requests for embedded objects, as many Web pages contain several or more embedded objects each of which may be located on a distinct server. While a slow but bursty stream of requests from a Web browser will eventually be released by the throttle, mail servers, Web crawlers, and other legitimate services that issue first-contact connections at a rate greater than once per second will overflow the queue. In this case, the virus throttling algorithm quarantines the host and allows no further first-contact connections.

To achieve rate limiting with a better false positive rate we once again present a solution inspired by sequential hypothesis testing and that relies on the observation that benign first-contact connections are likely to succeed whereas those issued by scanners are likely to fail. This credit-based approach, however, is unlike $\overleftarrow{HT}$ in that it assumes that a connection will fail until evidence proves otherwise. Because it does not wait for a timeouts to act, it can react immediately to a burst of connections and halt the flow so that $\overleftarrow{HT}$ can then make a more informed decision as to whether the host is infected. As it does not force connections to be evaluated in order, CBCRL can also immediately process evidence of connection successes. This will enable it to quickly increase the allowed first-contact connection rate when these requests are benign.

CBCRL, as summarized in Table 5.1, works by allocating to each local host, l, a starting balance of C_0 credits, which can be used for issuing first-contact connection requests. Whenever a first-contact connection request is observed, the sending host's balance is

decremented by p. If the successful acknowledgment of a first-contact connection is observed, the host that initiated the request is issued with q additional credits. No action is taken when connections fail, as the cost of issuing a first-contact connection has already been deducted from the issuing host's balance. Finally, first-contact connection requests are blocked if the host does not have any credit available ($C_l = 0$).

Table 5.1: The underlying equations behind credit-based connection rate limiting. Changes to a host's balance are triggered by the first-contact connections (FCCs) it initiates and by the passing of time.

Event	Change to C_l
Starting balance	$C_l \leftarrow C_0$
FCC issued by l	$C_l \leftarrow C_l - p$
FCC succeeds	$C_l \leftarrow C_l + q \ (q > p)$
Every second	$C_l \leftarrow \max(C_0, rC_l)$ if $C_l > C_0 \ (0 < r < 1)$
Allowance	$C_l \leftarrow 1$ if $C_l = 0$ for t seconds

If a first-contact connection succeeds with probability θ, its expected payoff from issuing that connection is its expected success credit minus its cost, or $q\theta - p$. This payoff is positive for $\theta > \frac{p}{q}$ and negative otherwise. To penalize a host with a less than 50% success rate, we set $q = 2$ and $p = 1$. Hosts that scan with a low rate of successful connections ($< \frac{p}{q}$) will quickly consume their credits whereas benign hosts that issue first-contact connections with high rates of success will increase their credits each time they invest them.

As described so far, the algorithm could result in two undesirable states. First, a host could acquire a large number of credits while performing a benign activity (e.g., Web crawling) which could be used later by a scanning worm. Second, a network outage could cause a benign host to use all of its credits after which it would starve for a lack of first-contact connection successes.

These problems are addressed by providing each host with a small allowance and by putting in place a high rate of inflation. If a host has been without credits for t seconds, we issue the host a single credit ($C_l \leftarrow 1$ if $C_l \leq 0$). This not only ensures that the host does not starve, but enables us to collect another observation to feed into the hypothesis test ($\overleftarrow{HT}$).

Because $\overleftarrow{HT}$, as configured below in Section 5.3, observes all first-contact connection requests as successes or failures within three seconds, providing a starving process with a credit allowance only after more than three seconds have passed ensures that $\overleftarrow{HT}$ will have been executed on all previously issued first-contact connection requests. In other words, t should be set such that it is greater than a timeout value based on which $\overleftarrow{HT}$ determines a failure for a non-responding first-contact connection request. If $\overleftarrow{HT}$ has already concluded that the host is a worm, it is expected that the system will be quarantined and so no requests will reach their destination regardless of the credit balance.

For each second that passes, a host that has acquired more than C_0 credits will be forced to surrender up to $(1 - r)$ of them, but not so many as to take its balance below C_0 ($C_l \leftarrow$

$\max(C_0, rC_l)$ if $C_l > C_0$). A host that is subject to the maximum inflation rate, with a first-contact connection rate γ, success rate $\theta > 0$, and credit balance $C_{l,t}$ at time t, will see this balance reach an equilibrium state $\hat{C}$ when $\hat{C} = C_{l,t} = C_{l,t+1}$.

$$\begin{aligned} C_{l,t+1} &= r[C_{l,t} + \gamma \cdot (q\theta - p)] \\ \hat{C} &= r[\hat{C} + \gamma \cdot (q\theta - p)] \\ \hat{C} &= \frac{r}{1-r} \cdot \gamma \cdot (q\theta - p) \end{aligned}$$

One way to choose the inflation constant r is to ensure that, in the upcoming second, a host that has a perfect first-contact connection success rate ($\theta = 1$) will have twice as many credits as it could have needed in the previous second. In the case of $p = 1$ and $q = 2$:

$$\begin{aligned} \frac{r}{1-r} \cdot \gamma &= 2 \cdot \gamma \\ r &= \frac{2}{3} \end{aligned}$$

Also note that the maximum inflation rate, which seems quite steep, is only fully applied when $\hat{C} \geq 15$, which in turn occurs only when the first-contact connection rate r is greater than 7.5 requests per second. Twycross and Williamson's virus throttle, on the other hand, can only assume that any host with a first-contact connection rate consistently greater than 1 per second is a worm.

The constant of 10 was chosen for the starting credit balance (and for the equilibrium minimum credit balance for benign hosts with first-contact connection rates below 5 requests/second) in order to match the requirements of our sequential hypothesis test ($\overleftarrow{HT}$) as currently configured (see parameters in Section 5.3), which itself requires a minimum of 10 observations in order to conclude that a host is engaged in scanning. Slowing the rate at which the first 10 observations can be obtained will only delay the time required by $\overleftarrow{HT}$ to conclude that a host is engaged in scanning. Should the parameters of $\overleftarrow{HT}$ be reconfigured and the minimum number of observations required to conclude a host is a scanner change, the starting credit balance for rate-limiting can be changed to match it.

5.3 Experimental Setup

We evaluated our algorithms using two traces collected at the peering link of a medium sized ISP: one collected in April 2003 (`isp-03`) containing 404 active hosts and the other in January 2004 (`isp-04`) containing 451 active hosts. These traces, summarized in Table 5.2, were collected using `tcpdump`.

Obtaining usable traces was difficult. Due to privacy concerns, network administrators are loathe to share any traces, let alone those that contain payload data in addition to headers. Yet we required the payload data in order to manually determine which, if any, worm was present on a host that was flagged as infected.

Table 5.2: Summary of network traces

	`isp-03`	`isp-04`
Date	2003/04/10	2004/01/28
Duration	627 minutes	66 minutes
Total outbound connection attempts	1,402,178	178,518
Total active local host	404	451

In configuring the reverse sequential hypothesis test ($\overleftarrow{HT}$), first-contact connection requests were interpreted as failures if they were not acknowledged within a three second grace period. First-contact connection requests for which TCP RST packets were received in response were immediately reported as failure observations. Connection success probability estimates were chosen to be:

$$\theta_0 = 0.7 \qquad \theta_1 = 0.1$$

Confidence requirements were set to:

$$\alpha = 0.00005 \qquad \beta = 0.99$$

Note that these confidence requirements are for each reverse sequential hypothesis test, and that a test is performed for each first-contact connection that is observed. Therefore, the false positive rate (α) is chosen to be particularly low as testing will occur many times for each host.

For each local host we maintained a Previously Contacted Host (PCH) set of only the last 64 destination addresses that each local host had communicated with (LRU replacement). In the experiment, a first-contact connection request was any TCP SYN packet or UDP packet addressed to a host that was not in the local host's PCH set. While using a fixed sized PCH set demonstrates the efficacy of our test under the memory constraints that are likely to occur when observing large (e.g., class B) networks, this fixed memory usage comes at a cost. As described in Section 5.5, it is possible for a worm to exploit limitations in the PCH set size to avoid having its scans detected.

For comparison, we also implemented Twycross and Williamson's virus throttle as described in [76]. Because our traces contain only those packets seen at the peering point, our results may differ from a virus throttle implemented at each local host as Twycross and Williamson recommend. However, because observing connections farther from the host results in a reduction in the number of connections observed, it should only act to reduce the reported number of false positives in which benign behavior is throttled.

All algorithms were implemented in `Perl`. We used traces that had been pre-processed by the Bro Network Intrusion Detection System [1, 52].

We did not process FTP data-transfer, finger, and ident connections as these connections are the result of local hosts responding to remote hosts, and are not likely to be accepted by a host that has not issued a request for such a connection. These connections are thus unlikely to be useful for worm propagation.

5.4 Results

$\overleftarrow{HT}$ detected two hosts infected with Code Red II [13, 69] from the April, 2003 trace (`isp-03`). Our test detected one host infected with Blaster [24], three hosts infected with MyDoom [26, 85], and one host infected with Mimail.j [25] from the January, 2004 trace (`isp-04`). The worms were conclusively identified by painstakingly comparing the logged traffic with the cited worm descriptions at various online virus/worm information libraries. Our test also identified four additional hosts that we classify as HTTP scanners because each sent SYN packets to port 80 of at least 290 addresses within a single class B network. These results are summarized in Table 5.3.

Table 5.3: Alarms reported by reverse sequential hypothesis testing combined with credit-based rate limiting. The cause of each alarm was later identified manually by comparing observed traffic to signature behaviors described at online virus libraries.

	`isp-03`	`isp-04`
Worms/Scanners detected		
`Code Red II`	2	0
`Blaster`	0	1
`MyDoom`	0	3
`Mimail.j`	0	1
HTTP (other)	3	1
Total	5	6
False alarms		
HTTP	0	3
SMTP	0	3
Total	0	6
P2P detected	6	11
Total identified	11	23

While peer-to-peer applications are not necessarily malicious, many network administrators do not classify them as benign. Peer-to-peer file sharing applications also exhibit ambiguous network behavior, as they attempt to contact a large number of transient peers that are often unwilling or unavailable to respond to connection requests. While peer-to-peer clients are deemed undesirable on most of the corporate networks that we envision our approach being used to protect, it would be unfair to classify these hosts as infected. For this reason we place hosts that we detect running peer-to-peer applications into their own category. Networks that allow peer-to-peer traffic may avoid these false alarms by limiting the detection algorithm to outgoing traffic to the well known port numbers [75]. However, this may not eliminate false alarms due to peer-to-peer applications because some of them tunnel through port 80.

Three additional false alarms were reported for three of the 60 (`isp-04`) total hosts transmitting SMTP traffic. We suspect the false alarms are the result of bulk retransmission of those emails that have previously failed when the recipients' mail servers were unreachable.

We suggest that organizations may want to white-list their SMTP servers, or significantly increase the detection thresholds for SMTP.

The remaining three false alarms are specific to the `isp-04` trace, and resulted from HTTP traffic. It appears that these false alarms were raised because of a temporary outage at a destination network at which multiple remote hosts became unresponsive. These may have included servers used to serve embedded objects.

Upon discovering these failures, we came to realize that it would be possible for an adversary to create Web sites that served pages with large numbers of embedded image tags linked to non-responsive addresses. If embedded with scripts, these sites might even be designed to perform scanning of the client's network from the server. Regardless, any client visiting such a site would appear to be engaged in HTTP scanning. To prevent such denial of service attacks that fool a user to generating scan-like traffic, we require a mechanism for enabling users to deactivate quarantines triggered by HTTP requests. We propose that HTTP requests from such hosts be redirected to a site that uses a CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart [79]), to confirm that a user is present and was using a Web browser at the time of quarantine.

Table 5.4: Alarms reported by virus throttling [76]

	`isp-03`	`isp-04`
Worms/Scanners detected		
`Code Red II`	2	0
`MyDoom`	0	1
HTTP (other)	1	1
Total	3	2
False alarms	0	0
P2P detected	2	3
Total identified	5	5

Results for our implementation of Twycross and Williamson's virus throttle [76] are summarized in Table 5.4. Their algorithm blocked both instances of Code Red II, but failed to detect Blaster, three instances of MyDoom (which is admittedly an email virus and not an IP scanning worm), and two low rate HTTP scanners. It did, however, detect one host infected with MyDoom that $\overleftarrow{HT}$ failed to detect. The MyDoom infected host was also active in accessing stock trading Web sites while transmiting a viral email to other hosts. This combining activity resulted in a high first-contact connection rate, and thus triggered the virus trottle. But, because of a low failure rate due to accesses to legitimate Web sites, it did not trigger $\overleftarrow{HT}$. The virus throttle also detected fewer hosts running peer-to-peer applications, which for fairness we classify as a reduction in false alarms in virus throttling's favor in our composite results summarized in Table 5.5.

These composite results for both traces report the number of hosts that resulted in alarms and the number of those alarms that were detections of the 12 worms located in our traces.

Table 5.5: Composite results for both traces: A total of 7 HTTP scanning worms and 5 email worms were present.

	Alarms	Detection	Efficiency	Effectiveness
$\overleftarrow{HT}$	34	11	0.324	0.917
virus-throttling	10	5	0.500	0.417

We also include the *efficiency*, which is the number of detections over the total number of alarms, and the *effectiveness*, which is the total number of detections over the total number of infected hosts we have found in these traces. While $\overleftarrow{HT}$ is somewhat less efficient than virus throttling, we believe that the more-than-two-fold increase in effectiveness is well worth the trade-off. In addition, corporate networks that forbid peer-to-peer file sharing applications will see a two-fold increase in efficiency.

Table 5.6 shows the number of hosts that had connection requests blocked by the CBCRL algorithm and the number of hosts that were rate-limited by Twycross and Williamson's algorithm. For credit-based connection rate limiting, we say that a machine has been rate limited if a single packet is dropped. For the virus throttle, we say that a machine has been rate limited if the outgoing delay queue length is greater than five, giving Twycross and Williamson the benefit of the doubt that users won't notice unless connections are severely throttled. CBCRL only limited the rates of hosts that our reverse sequential hypothesis test reported as infected. In contrast, even given our generous definition, more than 10% of the hosts in both traces were rate limited by Twycross and Williamson's algorithm.

Table 5.6: Comparison of rate limiting by CBCRL vs. virus throttling

	CBCRL		Virus Throttling	
	`isp-03`	`isp-04`	`isp-03`	`isp-04`
Worms/Scanners	5	1	3	4
P2P	4	8	3	7
Unnecessary rate limiting	0	0	84	59

Table 5.7 reports the number of first-contact connections permitted by the two approaches for those scanners that both detected. Code Red II is a fast scanner, and so virus throttling excels in blocking it after 6 to 7 connection requests. This speed is expected to come at the price of detecting any service, malicious or benign, that issues high-rate first-contact connections.

$\overleftarrow{HT}$ with credit-based connection rate limiting detects worms after a somewhat higher number of first-contact connections are permitted (10), but does so regardless of the scanning rate. Whereas our approach detects a slow HTTP scanner after 10 first-contact connection requests, the virus throttle requires as many as 526.

Table 5.7: The number of first-contact connections permitted before hosts were reported as infected: the value pairs represent individual results for two different Code Red II infections and two different HTTP scanners.

	$\overleftarrow{HT}$ with CBCRL	Virus Throttling
`Code Red II`	10,10	6,7
Other HTTP scanners	10,10	102,526

5.5 Limitations

CBCRL is resilient to network uplink outages as hosts starved for credits will receive an allowance credit seconds after the network is repaired. Unfortunately, this will be of little consolation as Reverse Sequential Hypothesis Testing ($\overleftarrow{HT}$) may have already concluded that all hosts are scanners. This may not be a problem if network administrators are given the power to invalidate observations made during the outage period, and to automatically reverse any quarantining decisions that would not have been taken without these invalid observations.

Of greater concern is that both $\overleftarrow{HT}$ and CBCRL rely exclusively on the observation that hosts engaged in scanning will have lower first-contact connection success rates than benign hosts. New hypotheses and tests are required to detect worms for which this statistical relationship does not hold.

Also problematic is that two instances of a worm on different networks could collaborate to ensure that none of their first-contact connections will appear to fail. For example, if worm A does not receive a response to a first-contact connection request after half the timeout period, it could send a message to worm B asking it to forge a connection response. This *forged response attack* prevents our system from detecting connection failures. To thwart this attack for TCP connections, a worm detection system implemented on a router can modify the TCP sequence numbers of traffic as it enters and leaves the network. For example, the result of a hash function $h(\text{IP}_{\text{local}}, \text{IP}_{\text{remote}}, \text{salt})$ may be added to all sequence numbers on outgoing traffic and subtracted from all incoming sequence numbers. The use of the secret salt prevents the infected hosts from calculating the sequence number used to respond to a connection request which they have sent, but not received. By storing the correct sequence number in the FCC queue, responses can then be validated by the worm detection system. However, modifying sequence numbers can disrupt a pre-existing connection that is already in progress when the detection system starts running. It is important that the detection system modifies a packet's sequence number (or acknowledgement number) only if the packet belongs to a connection that is initiated by a local host and the connection's initial SYN packet has been observed by the detection system.

Another concern is the possibility that a worm could arrive at its target already in possession of a list of known repliers – hosts that are known to reply to connection requests at a given port. This *known-replier attack* could employ lists that are programmed into the worm at creation, or accumulated by the worm as it spreads through the network. First-contact connections to these known-repliers will be very likely to succeed and can be interleaved with

scans to raise the first-contact connection success rate. A one to one interleaving is likely to ensure that more than half of all connections succeed. This success rate would enable the scanner to bypass credit-based connection rate limiting, and delay detection by Reverse Sequential Hypothesis Testing until the scanner had contacted all of its known-repliers. What's worse, a worm could avoid detection altogether if the detection system defines a first-contact connection with respect to a fixed sized previously contact host (PCH) set. If the PCH set tracks only the n previously visited hosts, the scanner can cycle through $(n/2)+1$ known-repliers, interleaved with as many new addresses, and *never* be detected.[4] To prevent a worm from scanning your local network by interleaving connections to known-repliers outside of your network, Weaver *et al.* [83] propose that one hypothesis test be run for local connections (i.e., those within the same IP block) and another for connections to remote hosts. If hosts in your local network are widely and randomly dispersed through a large IP space,[5] then a worm will have a low probability of finding another host to infect before being quarantined.

A worm might also avoid detection by interleaving scanning with other apparently benign behavior, such as Web crawling. A subset of these *benign interleaving attacks* can be prevented by detecting scanners based on the destination port they target in addition to the source IP of the local host. While it is still fairly easy to create benign looking traffic for ports such as HTTP, for which one connection can lead to information about other active hosts receptive to new connections, this is not the case for ports such as those used by SSH. Running separate scan detection tests for each destination port that a local host addresses can ensure that connections to one service aren't used to mask scans to other services.

Finally, if an infected host can impersonate other hosts, the host could escape quarantine and cause other (benign) hosts to be quarantined. To address these *address impersonation attacks*, it is important that a complete system for network quarantining include strong methods for preventing IP masquerading by its local hosts, such as switch-level egress filtering. Host quarantining should also be enforced as close to the host as is possible without relying on the host to quarantine itself. If these boundaries cannot be enforced between each host, one must assume that when one machine is infected, all of the machines within the same boundary will also be infected.

5.6 Summary

When combined, credit-based connection rate limiting and reverse sequential hypothesis testing ensure that worms are quickly identified with an attractively low false alarm rate. While no current system can detect all possible worms, our new approach is a significant improvement over prior methods, which detect a smaller range of scanners and unnecessarily delay network traffic. What's more, the techniques introduced in this work lend themselves to efficient implementation, as they need only be activated to observe a small

[4] For detecting such a worm, a random replacement policy will be superior to an LRU replacement policy, but will still not be effective enough for long known-replier lists.

[5] Randomly dispersing local hosts through a large IP space can be achieved by using a network address translation (NAT) switch.

subset of network events and require little calculation for the common case that traffic is benign.

As worm authors become aware of the limitations discussed in Section 5.5, it will be necessary to revise our algorithms to detect scanning at the resolution of the local host (source address) and targeted service (destination port), rather than looking at the source host alone. Solutions for managing the added memory requirements imposed by this approach have been explored by Weaver, Staniford, and Paxson [83].

The intrusiveness of credit-based connection rate limiting, which currently drops outgoing connection requests when credit balances reach zero, can be further reduced. Instead of halting outgoing TCP first-contact connection requests from hosts that do not maintain a positive credit balance, the requests can be sent immediately and the responses can be held until a positive credit balance is achieved. This improvement has the combined benefits of reducing the delays caused by false rate limiting while simultaneously ensuring that fewer connections are allowed to complete when a high-speed scanning worm issues a burst of connection requests. As a result, the remaining gap in response speed between credit-based connection rate limiting and Twycross and Williamson's virus throttle can be closed while further decreasing the risk of acting on false positives.

Finally, we can employ additional indicators of infection to further reduce the number of first-contact connection observations required to detect a worm. For example, it is reasonable to conclude that, when a host is deemed to be infected, those hosts to which it has most recently initiated successful connections are themselves more likely to be infected (as was the premise behind GrIDS [68]). We propose that this be accomplished by adding an event type, the report of an infection of a host that has recently contacted the current host, to our existing hypothesis test.

Chapter 6

Detection of Targeting Worm Propagations

The previous chapter presented algorithms for detecting network worms that use random scanning to discover new victims. This chapter describes an algorithm for detecting fast-propagating worms that use high-quality targeting information. It then presents a unified framework for detecting network worms independently of their target discovery strategy.

If a network worm penetrates a site's perimeter, it can quickly spread to other vulnerable hosts inside the site. The infection propagates by the compromised host repeatedly attempting to contact and infect new potential victims. Figure 6-1 illustrates a situation where a worm bypasses a firewall and then propagates to local machines via an infected laptop plugged in to a local network. The traffic pattern of fast worm propagation—a single host quickly contacting many different hosts—is a prominent feature across a number of types of worms, and detecting such patterns constitutes the basis for several worm detection approaches [14, 76].

The problem of accurately detecting such worm propagation becomes particularly acute for enterprise networks comprised of a variety of types of hosts running numerous, different applications. This diversity makes it difficult to tune existing worm detection methods [14,76] that presume preselected thresholds for connection rates and window sizes over which to compute whether a host's activity is "suspicious." First, finding a single threshold rate that accommodates most of benign hosts requires excessive tuning because of diverse application behaviors (e.g., a Web browser generating multiple concurrent connections to fetch embedded objects vs. an SSH client connecting to a server). Second, the window size chosen to compute the average rate affects the detection speed and accuracy; if too small, the detection algorithm is less resilient to small legitimate connection bursts, but if too big, the detection algorithm reacts slowly to fast propagating worms, for which prompt response is vital.

We first develop an algorithm for detecting fast-propagating worms that use high-quality *targeting* information. We base our approach on analyzing the rate at which hosts initiate connections to new destinations. One such class of worms are those that spread in a

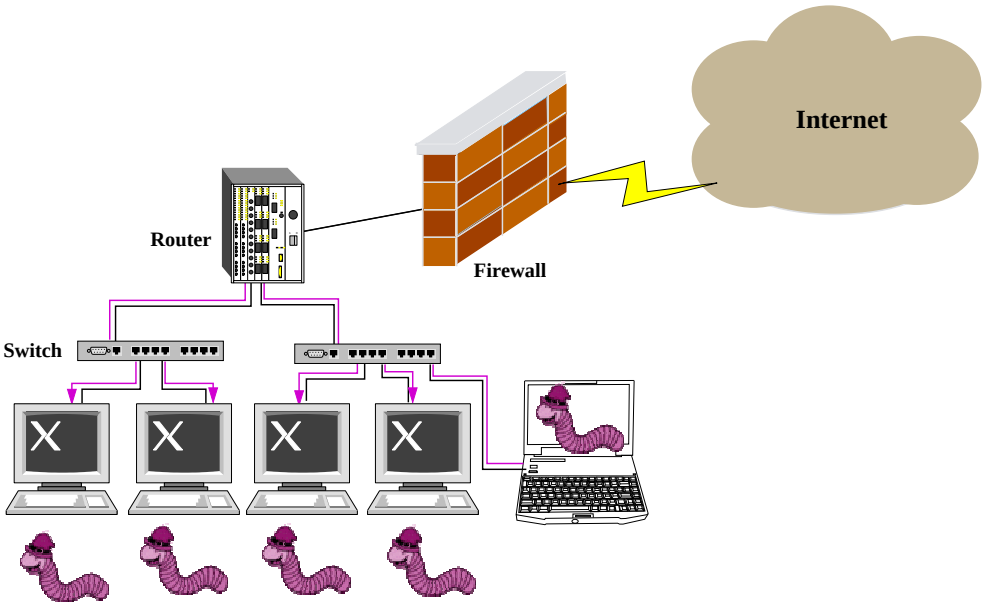


Figure 6-1: Worm propagation inside a site

topological fashion [67, 82]: they gather information on the locally infected host regarding other likely victims. For example, the Morris worm examined .rhosts files to see what other machines were known to the local machine [20, 63]. A related technique is the use of meta-servers, such as worms that query search engines for likely victims [22]. These targeting worms can spread extremely quickly, even using relatively low-rate scanning, because the vulnerability density of the addresses they probe is so much higher than if they use random scanning. Furthermore, these worms can evade many existing worm defense systems that rely on the artifacts of random scanning such as number of failed connections and the absence of preceding DNS lookups [14, 83, 84].

Our detection algorithm, Rate-Based Sequential Hypothesis Testing (RBS), operates on a per-host and per-connection basis and does not require access to packet contents. It is built on a probabilistic model that captures benign network characteristics, which enables us to discriminate between benign traffic and worm traffic. RBS also provides an analytic framework that enables a site to tailor its operation to its network traffic pattern and security policies.

We then present RBS + TRW, a unified framework for detecting fast-propagating worms independent of their target discovery strategy. RBS + TRW is a blend of RBS and the Threshold Random Walk (TRW) algorithm from Chapter 4, which rapidly discriminates between random scanners and legitimate traffic based on their differing rates of connection failures.

Section 6.1 presents an analysis of network traces we obtained from two internal routers of Lawrence Berkeley National Laboratory (LBL). Such data allow us to assess RBS's efficacy in detecting worms that remain inside an organization network, rather than just those that manifest in a site's external Internet traffic (a limitation of previous studies).

The traced traffic includes more than 650 internal hosts, about 10% of the total at the site. We examine the distribution of the time between consecutive *first-contact connection requests*, defined in Chapter 5 as a packet addressed to a host with which the sender has not recently communicated. Our analysis finds that for benign network traffic, these interarrival times are bursty, but within the bursts can be approximately modeled using exponential distributions with a few hundred millisecond average intervals.

In Section 6.2, we develop the RBS algorithm, based on the same sequential hypothesis testing framework as TRW. RBS quickly identifies hosts that initiate first-contact connection requests at a rate n times higher than that of a typical benign host. RBS updates its decision process upon each data arrival, triggering an alarm after having observed enough empirical data to make a distinction between the candidate models of (somewhat slower) benign and (somewhat faster) malicious host activity.

In Section 6.3, we evaluate RBS using trace-driven simulations. We find that when n is small, RBS requires more empirical data to arrive at a detection decision; for example, it requires on average 10.4 first-contact connections when $n = 5$. However, when n is larger, RBS provides accurate and fast detection. On the other hand, we show that a fixed-threshold rate-based scheme will inevitably require more difficult tradeoffs between false positives and false negatives.

Section 6.4 presents RBS + TRW, which automatically adapts between the rate at which a host initiates first-contact connection requests and observations of the success of these attempts, combining two different types of worm detection. Using datasets that contain active worms caught in action, we show that RBS + TRW provides fast detection of two hosts infected by Code Red II worms, while generating less than 1 false alarm per hour.

6.1 Data Analysis

We hypothesize that we can bound a benign host's network activity by a reasonably low fan-out per unit time, where we define fan-out as the number of first-contact connection requests a given host initiates. This fan-out per unit time, or *fan-out rate*, is an important traffic measure that we hope will allow us to separate benign hosts from relatively slowly scanning worms. In this section, we analyze traces of a site's internal network traffic, finding that a benign host's fan-out rate rarely exceeds a few first-contact connections per second, and time intervals between those connections can be approximately modeled as exponentially distributed.

We analyze a set of 22 anonymized network traces, each comprised of 10 minutes' of traffic recorded at `LBL` on Oct. 4, 2004. These were traced using `tcpdump` at two internal routers within `LBL`, enabling them to collect bidirectional traffic originated by internal hosts to both external hosts outside `LBL` and to other internal hosts inside `LBL`. While ideally we would have such traces from a number of different sites in order to assess the range of behavior normally seen, such traces have to date been unavailable. Indeed, we believe the internal traces to which we have access are unique or nearly so for the research community at present. Thus, we view them as highly valuable, if fundamentally limited, though we need to continually keep in mind the caution that we should not readily generalize from

them. (A much larger dataset, LBL-II, later became available from this same site. We use it in Section 6.4 to evaluate RBS + TRW, independently of data analysis.)

Table 6.1 summarizes the LBL dataset after some initial filtering to remove periodic NTP traffic and "triggered" connections in which a connection incoming to a host causes the host to initiate a secondary connection outbound. Such triggered connections should not be considered first-contact connections when assessing whether a host is probing.

The table shows that the traffic between internal LBL hosts consists of about 70% of the total outbound traffic recorded in the datasets. Had we traced the traffic at the site's border, we would have seen much less of the total network activity, and lower first-contact connections accordingly.

Table 6.1: LBL dataset summary: This analysis does not include NTP traffic or triggered outgoing connections such as Ident, Finger, and FTP data-transfer.

Outgoing connections	
to internal hosts	32,967
to external hosts	16,082
total	49,049
Local hosts	≥ 652

For each 10-minute trace, we observe that the number of active internal hosts that initiated any outbound traffic during the observation period varies. The last row in Table 6.1 shows that the largest number of active internal hosts in a 10-minute trace is 652. Because each trace was anonymized separately, we are unable to tell how many distinct internal hosts appear across all of the traces.

We plot the cumulative distribution of per-host fan-out in Figure 6-2. We see that over 99.5% of hosts contacted fewer than 60 different hosts in 10 minutes, which results in less than 0.1/sec fan-out rate on average. However, the top 0.5% most active hosts greatly stretch out the tail of the distribution. In what follows, we examine those hosts with a high fan-out rate to understand what distinguishes their behavior from that of worm propagation. Then, we find a set of "purely" benign hosts, which we use to develop a model that captures their fan-out rate statistics.

6.1.1 Separating Benign Hosts

Our starting point is the assumption that a host is benign if its fan-out rate is less than 0.1/sec averaged over a 10-minute monitoring period. Note that Twycross and Williamson [76] use a 1/sec fan-out rate as a maximum allowed speed for throttling virus spreads. Only 9 hosts exceed this threshold. Of these, 4 were aliases (introduced by each trace having a separate anonymization namespace) for an internal scanner used by the site for its own vulnerability assessment. Of the remainder, 3 hosts are mail servers that forward large volumes of email, and the other 2 hosts are internal web crawlers that build search engine databases of the content served by internal Web servers.

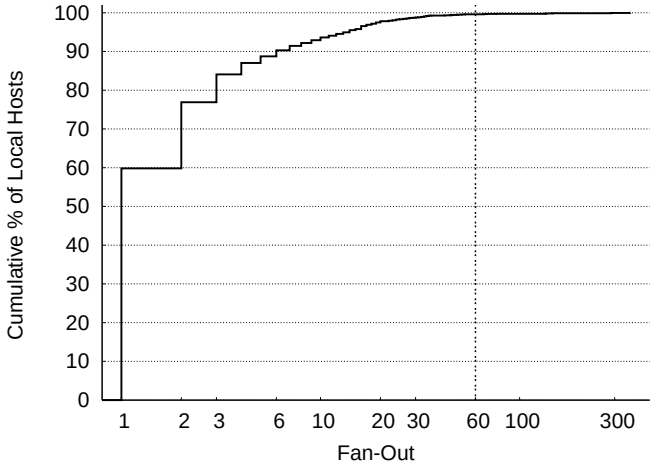


Figure 6-2: Fan-out distribution of an internal host's outbound network traffic for a 10 minute observation period: only 9 hosts contacted more than 60 different hosts in 10 minutes.

By manual inspection, we also later found another appearance of the internal scanner that we missed using our 0.1/sec fan-out rate threshold, as in that instance the scanner contacted only 51 different IP addresses during the 10-minute period. Table 6.2 shows the average fan-out per each type of scanners detected from the LBL dataset. Note that we do not include the mail servers here, as they are not scanners per se, but rather applications that happen in this environment to exhibit high fan-out.

Table 6.2: Scanners detected from the LBL dataset

Type	Count	Average fan-out
Internal scanner	5	196.4
Internal crawler	2	65.5

We exclude the scanners from our subsequent analysis, because including them would greatly skew the fan-out statistics of benign hosts. Given their low number, it is reasonable to presume that sites could maintain white-lists of such hosts to filter out detections of their activities by our algorithm.

6.1.2 Time Interval to Visit New Destinations

A host engaged in scanning or worm propagation will generally probe a significant number of hosts in a short time period, yielding an elevated first-contact connection rate. In this section, we analyze our dataset to determine the distribution of first-contact connection's interarrivals as initiated by benign hosts. We then explore the discriminating power of this metric for a worm whose interarrivals arrive a factor of n more quickly.

Figure 6-3 shows the distribution of the amount of time between first-contact connections for individual hosts. Here we have separated out the scanners (identified as discussed

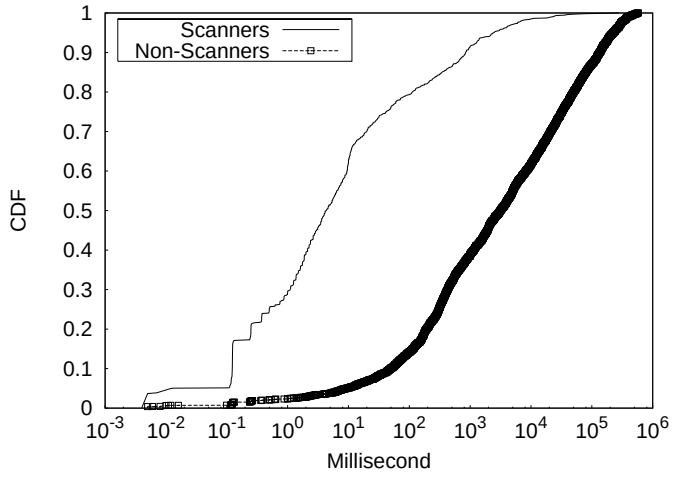


Figure 6-3: Distribution of first-contact interarrival time, per host

above), listing two groups, `scanners` and `non-scanners`. We see that scanners have a much shorter average interarrival time (1.1 sec) compared to the non-scanners (39.2 sec). Yet, the average is deceptive because of the uneven distribution of time intervals. Although the average non-scanner interarrival time is 39.2 sec, we often see benign, non-scanner hosts initiating multiple first-contact connections separated by very little (< 1 sec) time. In fact, these short time intervals account for about 40% of the total intervals generated by benign hosts, which makes it impractical to use 1/sec fan-out rate to identify possible worm propagation activity.

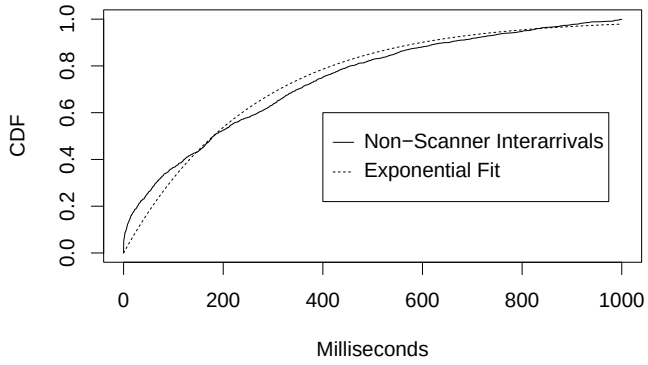


Figure 6-4: First-contact interarrivals initiated by benign hosts roughly follow an exponential distribution with mean $\mu = 261$ msec.

However, when focusing on sub-second interarrivals, we find that a benign host's short-time-scale activity fits fairly well to an exponential distribution, as illustrated in Figure 6-4. Here the fit to `non-scanners` uses $\mu = 261$ msec. In comparison, `scanners` on av-

erage wait no more than 69 msec between first-contact connections in this trace. We note that a scanner (or a worm) could craft its probing scheduling such that its fine-grained scanning behavior matches that of benign users, or at least runs slower than what we model as benign activity. However, this will significantly slow down the scanning speed, so compelling attackers to make this modification constitutes an advance in the ongoing "arms race" between attackers and defenders.

We also note that we could extract significantly more precise interarrival models—including differing mean interarrival rates—if we partitioned the traffic based on its application protocol. While investigating this refinement remains a topic for future work, in our present effort we want to explore the efficacy of as *simple* a model as possible. If our algorithm can prove effective without having to characterize different protocols separately, we will benefit a great deal from having fewer parameters that need to be tuned operationally.

In the next section, based on these characteristics of benign activity, we develop our detection algorithm, RBS, for quickly identifying scanners or worm infectees with a high accuracy.

6.2 Rate-Based Sequential Hypothesis Testing

In the previous section we examined traces and found that benign hosts often initiate more than one first-contact connection request per second, but in such cases we can approximate the interval between these connections with an exponential distribution. In this section, we develop a rate-based sequential hypothesis testing algorithm, RBS, which aims to quickly identify hosts issuing first-contact connections at rates higher than what we model as benign activity.

Let H_1 be the hypothesis that a given host is engaged in worm propagation, and let H_0 be the null hypothesis that the host exhibits benign network activity. A host generates an *event* when it initiates a connection to a destination with which the host has not previously communicated (since the observation began), i.e., when the host initiates a first-contact connection. We assume that the interarrival times of such events follow an exponential distribution with mean $1/\lambda_0$ (benign host) or $1/\lambda_1$ (scanner or a worm infectee). When a host generates the i^{th} event at time t_i, we can compute an interarrival time, $X_i = t_i - t_{i-1}$ for $i \geq 1$ and t_0 the initial starting point, and update the likelihood ratio of the host being engaged in scanning (or benign).

Define X_1, X_2, ..., X_n as a sequence of such interarrival times. Since each X_i follows an exponential distribution, $T_n = X_1 + X_2 + \cdots + X_n$ can be modeled as an n-Erlang distribution,

$$f_n(T_n|H_1) = \frac{\lambda_1(\lambda_1 T_n)^{n-1}}{(n-1)!} e^{-\lambda_1 T_n} \tag{6.1}$$

$$f_n(T_n|H_0) = \frac{\lambda_0(\lambda_0 T_n)^{n-1}}{(n-1)!} e^{-\lambda_0 T_n} \tag{6.2}$$

Based on Equations (6.1) and (6.2), we can develop a sequential hypothesis test in which we define the likelihood ratio as:

$$\Lambda(n, T_n) = \frac{f_n(T_n|H_1)}{f_n(T_n|H_0)} = \left(\frac{\lambda_1}{\lambda_0}\right)^n e^{-(\lambda_1-\lambda_0)T_n} \tag{6.3}$$

and the detection rules as:

$$\text{Output} = \begin{cases} H_1 & \text{if } \Lambda(n, T_n) \geq \eta_1 \\ H_0 & \text{if } \Lambda(n, T_n) \leq \eta_0 \\ \text{Pending} & \text{if } \eta_0 < \Lambda(n, T_n) < \eta_1 \end{cases}$$

where we can set η_1 and η_0 in terms of a target false positive rate, α, and a target detection rate, as discussed in Chapter 2.2.

$$\eta_1 \leftarrow \frac{\beta}{\alpha} \tag{6.4}$$

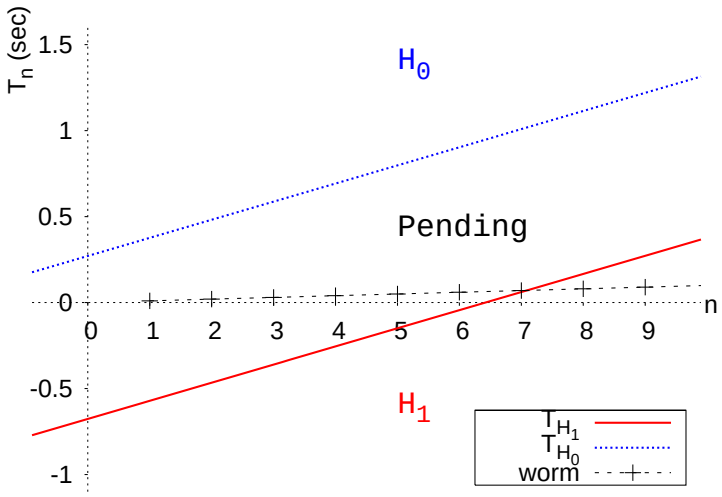
$$\eta_0 \leftarrow \frac{1-\beta}{1-\alpha} \tag{6.5}$$

As shown by Wald, above thresholds guarantee that the resulting false positive rate is bounded by $\frac{\alpha}{\beta}$ and the false negative rate is by $\frac{1-\beta}{1-\alpha}$ [80]. Given that β is usually set to a value higher than 0.99 and α to a value lower than 0.001, the margin of error becomes negligible (i.e., $\frac{1}{\beta} \approx 1$ and $\frac{1}{1-\alpha} \approx 1$).

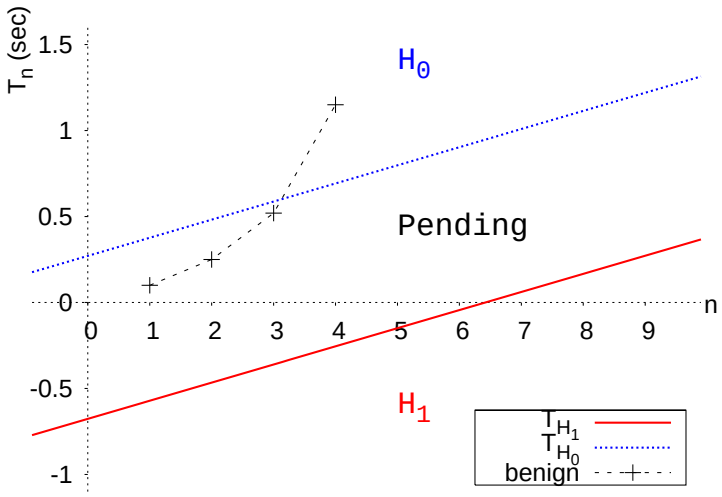
There are four parameters that need to be set in order to run RBS: α and β can be chosen with the values of the false positive rate (α) and the detection rate (β) that we want to achieve with the detection algorithm. However, the selection of two *priors*, λ_0 and λ_1, is less straightforward than that of α and β as it depends on an average fan-out rate of benign hosts (λ_0) at the site and the site's security policy (i.e., a host issuing first-contact connections at a rate higher than $n\lambda_0$ (λ_1) will trigger an alarm). In what follows, we show the impact of these four parameters on the detection result, which in turn sheds light on how they should be chosen.

Upon each arrival of a first-contact connection by a host, RBS estimates whether the current behavior provides evidence strong enough to choose one hypothesis against the other. For instance, if a host has initiated n first-contact connections and the elapsed time for the n^{th} connection is T_n, RBS chooses H_1 (worm) only if the likelihood ratio $\Lambda(n, T_n)$ exceeds η_1.

Using Equations (6.3) and (6.4), we can obtain a threshold on the elapsed time, T_{H_1}, below



(a) Fast spreading worm with 100 first-contact connections/second will be detected by RBS at the 8^{th} connection attempt.



(b) Benign host with 4 first-contact connections/second will be declared "benign" by RBS at the 4^{th} connection attempt.

Figure 6-5: T_{H_1} and T_{H_0} when $\lambda_0 = 3$/sec, $\lambda_1 = 20$/sec, $\alpha = 10^{-5}$, and $\beta = 0.99$. The X axis represents the n^{th} event and Y axis represents the elapsed time for the n^{th} event.

which we arrive at an H_1 (worm) decision:

$$\begin{aligned}
\frac{\beta}{\alpha} &\leq \Lambda(n, T_n) \\
\frac{\beta}{\alpha} &\leq \left(\frac{\lambda_1}{\lambda_0}\right)^n e^{-(\lambda_1-\lambda_0)T_n} \\
\ln\frac{\beta}{\alpha} &\leq n\ln\frac{\lambda_1}{\lambda_0} - (\lambda_1-\lambda_0)T_n \\
T_n &\leq n\frac{\ln\frac{\lambda_1}{\lambda_0}}{\lambda_1-\lambda_0} - \frac{\ln\frac{\beta}{\alpha}}{\lambda_1-\lambda_0} = T_{H_1}
\end{aligned} \tag{6.6}$$

Likewise, we can obtain a threshold elapsed time T_{H_0}, above which we conclude H_0 (benign host):

$$T_{H_0} \quad = n\frac{\ln\frac{\lambda_1}{\lambda_0}}{\lambda_1-\lambda_0} - \frac{\ln\frac{1-\beta}{1-\alpha}}{\lambda_1-\lambda_0} \tag{6.7}$$

Figure 6-5 shows how those threshold elapsed times, T_{H_1} and T_{H_0}, partition the area into three decision regions—H_1, H_0, and `Pending`. Figure 6-5(a) illustrates T_n of a host issuing first-contact connections at 100/second. At the 8^{th} event, T_8 falls below T_{H_1}, which drives the likelihood ratio to reach the H_1 decision. Note that with the set of parameters used in Figure 6-5, RBS holds a decision until it sees at least 7 events ; this occurs because the elapsed time, T_n, is always greater than T_{H_1} up to $n = 6$. T_i is a non-negative, non-decreasing random variable and T_{H_1} becomes positive when $n > 6.1$, given λ_0 =3/sec, λ_1 =20/sec, $\alpha = 10^{-5}$, and $\beta = 0.99$. This initial holding period makes RBS robust against a small traffic burst. We can shorten this initial holding period, however, if we use a smaller β or larger α. In general, Equation (6.6) provides important insights into the priors and the performance of RBS. T_{H_1} is a function of n, taking a form of $g(n) = a(n-c)$, where $a = (\ln\frac{\lambda_1}{\lambda_0})/(\lambda_1-\lambda_0)$ and $c = (\ln\frac{\beta}{\alpha})/(\ln\frac{\lambda_1}{\lambda_0})$:

1. α and β affect only c, the minimum number of events required for detection. For fixed values of λ_1 and λ_0, lower values of α or higher values of β (i.e., greater accuracy in our decisions) let more initial connections escape before RBS declares H_1. One can shorten this initial holding period by increasing α or decreasing β. But we can only do so to a limited degree, as c needs to be greater than the size of bursty arrivals that we often observe from applications on benign hosts, in order to avoid excessive false alarms. Another way to prevent damage from those initially allowed connection attempts is to hold them at a switch until proved innocent as discussed in Section 5.6.

2. λ_0 and λ_1 determine a, the slope of T_{H_1} over n. The inverse of the slope gives the minimum connection rate that RBS can detect. Any host generating first-contact connections at a higher rate than λ_1 intercepts $g(n)$ with probability 1. There is a built-in robustness in this, because the slope is strictly larger than $\frac{1}{\lambda_1}$ (what we model as a worm), which follows from the inequality $\ln(x) < x-1, 0 < x < 1$ (Figure 6-6).

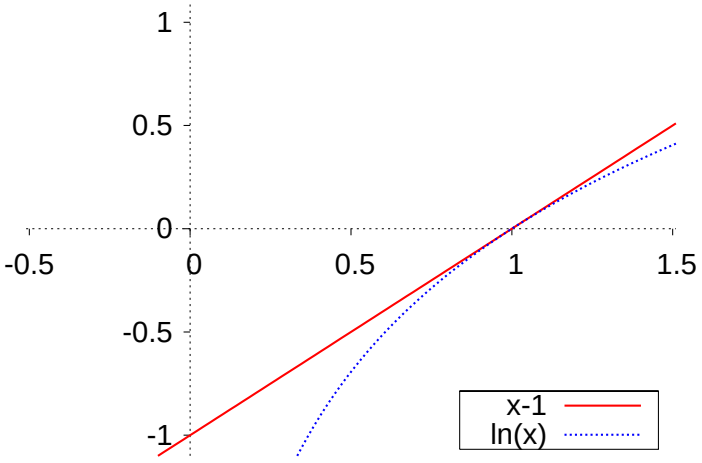


Figure 6-6: $\ln(x) < x - 1$ when $0 < x < 1$

3. Although we use λ_1 to model a worm's first-contact connection rate, RBS can detect any worm with a rate λ' provided that:

$$\lambda' > \frac{1}{a} = \frac{\lambda_1 - \lambda_0}{\ln \lambda_1 - \ln \lambda_0} \tag{6.8}$$

because a host with a rate higher than λ' will eventually cross the line of T_{H_1} and thus trigger an alarm.

6.3 Evaluation

We evaluate the performance of RBS in terms of false positives, false negatives, and the detection speed using a trace-driven simulation of the LBL dataset. The dataset contains 650 active hosts, including 14 hosts that are legitimate but exhibit network behavior resembling that of fast targeting worms. We then discuss the robustness of RBS to the bursty nature of benign traffic that a naïve fixed threshold based algorithm is unable to capture.

Each line in the LBL dataset represents a connection seen by the Bro NIDS [52], sorted by the timestamp of the first packet belonging to the connection. Connection information includes a source address, s, a destination address, d, and a connection start time. For each connection, our trace-driven simulator checks if s has previously accessed d. If not, it updates the likelihood of s being a scanner as described in Section 6.2.

In addition to the hosts identified in Table 6.2, by operating RBS we found 9 more hosts whose network activity involved some sort of scanning: 2 more instances of the internal scanner (not found using the previous simple fan-out threshold of 60 destinations over 10 minutes, because their scanning was limited to 7 and 34 destinations, respectively); a network monitoring system, "WhatsUp" [6], which contacted 14 hosts in 0.4 sec; and 6

Table 6.3: Trace-driven simulation results of RBS varying λ_1 when $\lambda_0 = 3.83$ Hz, $\alpha = 10^{-5}$, and $\beta = 0.99$: $\overline{N}|H_1$ represents the average number of first-contact connections initiated by the flagged hosts until detection. The final line gives the theoretical bound on the slowest scanner (or worm) the RBS can detect (Equation (6.8)).

$\lambda_1 =$	$2\lambda_0$	$3\lambda_0$	$4\lambda_0$	$5\lambda_0$	$10\lambda_0$	$15\lambda_0$	$20\lambda_0$	$25\lambda_0$
`scanners` (7)	4	5	6	6	7	7	7	7
`Web crawlers` (2)	2	2	2	2	2	2	2	2
`WhatsUp` (1)	0	1	1	1	1	1	1	1
`iprint` (6)	0	0	1	2	3	3	6	6
Total detection (16)	6	8	10	11	13	13	16	16
False positives	0	0	0	0	0	0	0	0
$\overline{N}\|H_1$	30.2	18.1	13.8	10.4	6.6	5.7	5.2	5.0
Theoretical bound (Hz)	> 5.5	> 7.0	> 8.3	> 9.5	> 15.0	> 19.8	> 24.3	> 28.5

instances of an `iprint` printer management client [5] that occasionally accesses all the printers to check availability. These all exhibit greater than 5/sec fan-out rates averaged over a second because of their bursty first-contact connections.

For high accuracy, we set $\beta = 0.99$ (99% target detection rate) and $\alpha = 0.00001$ (0.001% target false alarm rate). Note that α is set very low because the detection algorithm executes at every first-contact connection initiated by a local host, which adds up to a very large number of tests. We choose λ_0 such that $1/\lambda_0$ equals 261 (msec), the mean time interval to visit new destinations of benign hosts as shown in Section 6.1.2. However, there is no obvious pick for λ_1 since a worm can choose an arbitrary rate to propagate. If λ_1/λ_0 is close to 1, RBS takes longer to make a decision. It can also miss short bursts; but on the other hand, it can detect slower scanners than for higher λ_1/λ_0 ratios, per Equation (6.8).

Table 6.3 shows the simulation results of RBS for the `LBL` dataset as we vary λ_1 as a multiple of $\lambda_0 = 3.83$ Hz. With increasing λ_1, we see that RBS's detection rate increases without incurring false positives. Hosts that RBS often fails to detect include an internal scanner that probed only 7 hosts in a 10 minute trace, and 6 `iprint` hosts that accessed only 10 or fewer other printers, sometimes in two separate 5-connection bursts, which thins out the sources' average fan-out rate, making them difficult to detect.

Thus, while this assessment is against a fairly modest amount of data, we find the results promising. We conduct a more extensive evaluation in Section 6.4.

6.3.1 Limitations of Simple Rate-Base Thresholds

An issue to consider is whether we really need RBS's more sophisticated approach, or if a simpler scheme using a fixed-rate threshold suffices. We model such a simpler scheme as one that, upon each connection arrival, compares the fan-out rate, n/T, with a threshold η, alerting if the rate exceeds the threshold. Here, n is the number of first-contact connections from a host and T the elapsed time over which they occurred.

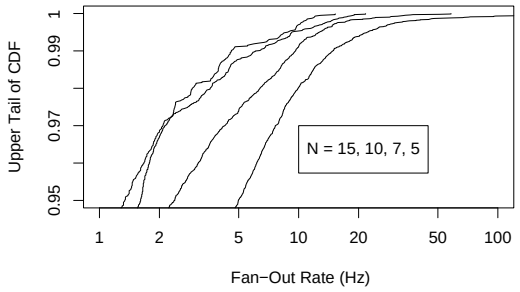


Figure 6-7: CDF of fan-out rates of non-scanner hosts using a window size of 15, 10, 7 and 5 (from left to right).

In this section we evaluate such schemes and find that they suffer from either significant false alarms, due to legitimate bursty connections, or significant false negatives. RBS is more robust to such bursts as it demands consistency over a larger number of observations before reaching a decision.

We compute a host's instantaneous fan-out rate as follows. For an outgoing connection initiated by the host at time t_c, we look back in time for the $n-1$ most recent first-contact connections. If the time of the first of these is t_p, then we calculate the fan-out rate as the ratio of $n/T = n/(t_c - t_p)$.

After removing the scanners listed in Table 6.3, Figure 6-7 shows the upper tail of the distribution of the fan-out rate of the remaining hosts, as a function of the aggregation window size n. Recall that any detection of these connections constitutes a false positive. So, for example, for windows of size $n = 7$, the 99th percentile occurs right around 10 Hz. Thus, using a window of size 7, to detect scanners that scan as slowly as 10 Hz, we must accept a false positive rate of 1% *per window*. With a window size of 7, this would mean over our dataset the detector would generate 118 false positives. While higher values of n reduce the false positive rate, they also will increase false negatives, such as the bursty scanners discussed in the previous section.

Comparing these curves with the slowest scanners detectable by RBS, per Table 6.3, we see that RBS gains significant power in both detecting slower or briefer scanners and in avoiding false positives. The essential advantage of RBS over the simpler schemes is that RBS effectively can *adapt* its window n and threshold η, rather than having to use single, fixed values for these.

6.4 RBS + TRW: A Combined Approach

RBS uses *fan-out rate* to differentiate benign traffic from scanners (or targeting worms), which we model as Poisson processes with rates λ_0 (benign) and λ_1 (scanner or a worm infectee), with $\lambda_0 < \lambda_1$. Another discriminatory metric proved to work well in detecting scanners (or random scanning worms) is the *failure ratio* of first-contact connections as discussed in Chapter 4, Chapter 5, and [83]. TRW works by modeling Bernoulli processes with **success** probabilities, θ_0 (benign) and θ_1 (scanner), with $1-\theta_0 < 1-\theta_1$. In this section, we develop a combined worm detection algorithm that exploits *both* a fan-out rate model and a failure ratio model. We evaluate the hybrid using trace-driven simulation, finding

that this combined algorithm, RBS + TRW, improves both overall accuracy and speed of detection.

Suppose that a given host has initiated connections to n different destinations, and that the elapsed time until the n^{th} connection is T_n. Among those n destinations, S_n peers accepted the connection request (success) and $F_n = n - S_n$ rejected (TCP RST) or did not respond (failure). Applying the models from RBS and TRW (Chapter 4), we obtain a conditional probability distribution function for worms:

$$\begin{aligned} f[(S_n, T_n)|H_1] &= P[S_n|T_n, H_1] \times f[T_n|H_1] \\ &= \binom{n}{S_n} \theta_1^{S_n} (1-\theta_1)^{F_n} \\ &\quad \times \frac{\lambda_1 (\lambda_1 T_n)^{n-1}}{(n-1)!} e^{-\lambda_1 T_n} \end{aligned}$$

where $P[S_n|T_n, H_1]$ is the probability of getting S_n success events when each event will succeed with an equal probability of θ_1, and $f[T_n|H_1]$ is an n-Erlang distribution in which each interarrival time is exponentially distributed with mean interarrival time $1/\lambda_1$.

Analogous to $f[(S_n, T_n)|H_1]$, for benign hosts we can derive:

$$\begin{aligned} f[(S_n, T)|H_0] &= \binom{n}{S_n} \theta_0^{S_n} (1-\theta_0)^{F_n} \\ &\quad \times \frac{\lambda_0 (\lambda_0 T_n)^{n-1}}{(n-1)!} e^{-\lambda_0 T_n}. \end{aligned}$$

We then define the likelihood ratio, $\Lambda(S_n, T_n)$, as

$$\begin{aligned} \Lambda(S_n, T_n) &= \frac{f[(S_n, T_n)|H_1]}{f[(S_n, T_n)|H_0]} \\ &= \left(\frac{\theta_1}{\theta_0}\right)^{S_n} \left(\frac{1-\theta_1}{1-\theta_0}\right)^{F_n} \\ &\quad \times \left(\frac{\lambda_1}{\lambda_0}\right)^{n} e^{-(\lambda_1 - \lambda_0) T_n}. \end{aligned}$$

It is interesting to note that $\Lambda(S_n, T_n)$ is just the product of Λ_{TRW} and Λ_{RBS}. Moreover, $\Lambda(S_n, T_n)$ reduces to Λ_{TRW} when there is no difference in fan-out rates between benign and scanning hosts ($\lambda_1 = \lambda_0$). Likewise, $\Lambda(S_n, T_n)$ reduces to Λ_{RBS} when there is no difference in success ratios ($\theta_1 = \theta_0$).

We evaluate this combined approach, RBS + TRW, using two new sets of traces, each of which contains different types of scanners and worm infectees that happen to wind up contrasting the strengths of RBS and TRW. We first categorize hosts into classes based on their fan-out rates and failure ratios. In what follows, we discuss types of scanners falling into each region and detection algorithms capable of detecting such hosts.

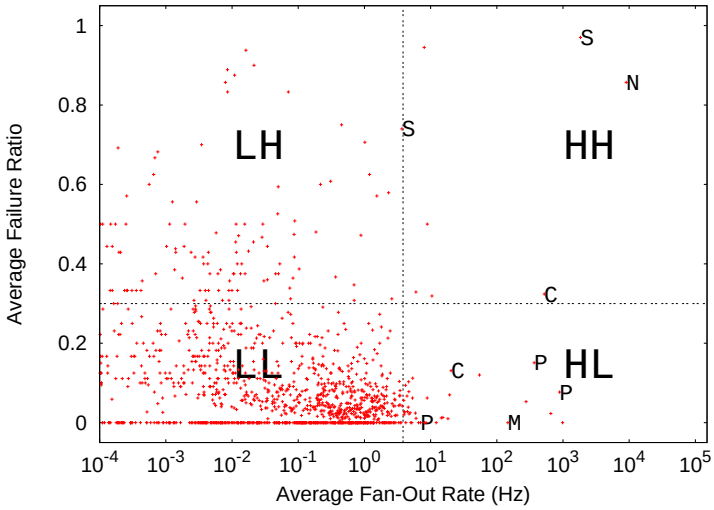
- **Class LH** (low fan-out rate, high failure ratio): Slow-scanning worms or scanners that probe at blindly (randomly or sequentially) will likely generate many failures, triggering TRW with a high probability.
- **Class HH** (high fan-out rate, high failure ratio): Fast-scanning worms (e.g., Code Red, Slammer) that exhibit both a high fan-out rate and a high failure ratio will very likely to drive both TRW and RBS to quickly reach their detection thresholds.
- **Class HL** (high fan-out rate, low failure ratio): Flash, metaserver, and topological worms [82] belong to this class. These worms build or acquire a list of target hosts and then propagate over only those potential victims, so their connection attempts tend to succeed. While these targeting worms can bypass TRW, their high fan-out rate should trigger RBS.
- **Class LL** (low fan-out rate, low failure ratio): Most benign hosts fall into this class, in which their network behavior is characterized by a low fan-out rate and a low failure ratio. Typically, a legitimate host's fan-out rate rarely exceeds a few first-contact connections per second. In addition, benign users do not initiate traffic to hosts unless there is reason to believe that the host will accept the connection request, and thus will yield a high success probability. Neither TRW nor RBS will trigger hosts in this class, which in turn, allows particularly stealthy worms, or passive "contagion" worms that rely on a user's behavior for propagation [82], to evade detection. Worms of this type represent a formidable challenge whose detection remains an open problem.

We use an average 3.83 Hz fan-out rate (λ_0) and 0.3 failure ratio ($1 - \theta_0$) as baselines in order to categorize hosts in our trace, where the setting for λ_0 comes from Section 6.1 and that for θ_0 from Section 5.3. We compute a fan-out rate with a sliding window of size 5 in order to capture bursty arrivals that often result from concurrent Web connections addressed to different Web sites for embedded objects. Figure 6-8 classifies hosts in the datasets based on the 3.83 Hz fan-out rate and 0.3 failure ratio thresholds.

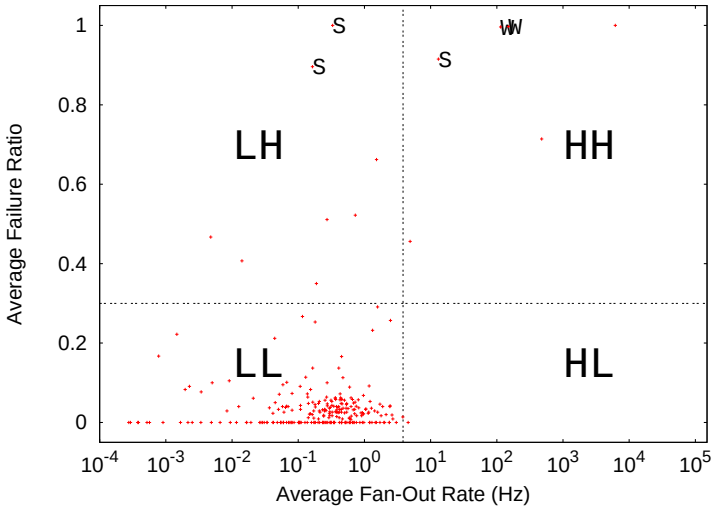
Table 6.4 shows the details of the datasets we use for evaluation. The `LBL-II` dataset was also collected from Lawrence Berkeley Laboratory (LBL). It is composed of 135 one-hour long traces from Dec. 2004 and Jan. 2005, recorded at internal routers connecting a variety of subnets to the rest of the laboratory and the Internet. The `ISP` dataset was recorded using `tcpdump` at the border of a small ISP in April 2003. It contains traffic from 389 active hosts during the 10-hour monitoring period (The high number of connections is due to worm infections during the time of measurement.).

The table shows the division of the internal hosts into the four categories discussed above. Manual inspection of the hosts in **HH**, **HL**, and **LH**[1] reveals that there are 9 (`LBL-II`) and 5 (`ISP`) hosts that are actual worms or scanners or whose behavior qualifies them as proxies for targeting worms that we aim to detect because of their high-fan-out behaviors: for `LBL-II`, the 3 **HH** hosts are one internal vulnerability scanner, one host that did a fast

[1]We looked into each host in those three classes for the `ISP` dataset, and the 66 of such hosts for the `LBL-II` dataset that generated more than 20 first-contact connections in a one-hour monitoring period.



(a) `LBL-II` (`S`: scanner, `N`: host running `nmap`, `M`: "Whatsup" monitor, `C`: internal Web crawler, `P`: print management host): There are 2 `S`'s, 1 `N`, 1 `M`, 2 `C`'s and 3`P`'s in the plot.



(b) `ISP` (`S`: scanner, `W`: Code Red II infectee): There are 3 `S`'s and 2 `W`'s in the plot. Two `W`'s are almost overlapped with each other.

Figure 6-8: Classification of hosts present in the evaluation datasets: Each point represents a local host that generated more than 5 first-contact connections.

Table 6.4: Evaluation datasets: `scanning` hosts include vulnerability scanners, worm infectees and hosts that we use proxies for targeting worms because of their anomalous high-fan-out rate.

			LBL-II	ISP
Outgoing Connections			796,049	1,402,178
Duration			137 hours	10.5 hours
H O S T S	**HH**	scanning	3	3
		benign	4	3
	LH	scanning	1	2
		benign	147	6
	HL	scanning	5	0
		benign	32	1
	LL	scanning	0	0
		benign	1195	255
	≤ 5 first-contact connections		2,621	119
	Total	scanning	9	5
		benign	3,999	384
		Total	4,008	389

`nmap` [4] scan of 7 other hosts, and one internal Web crawler that occasionally contacted tens of internal Web servers to update search engine databases; 1 **LH** host is another internal vulnerability scanner, whose average fan-out rate was 3.68 (slightly lower than the threshold); 5 **HL** hosts are 1 internal Web crawler, one "WhatsUp" monitor, and 3 printer management hosts. For `ISP`, the **HH** hosts were two Code Red II infectees plus an HTTP scanner, and the **LH** hosts were 2 slower HTTP scanners. We classify these 14 hosts as `scanning`.

The 4 **HH** hosts in the `LBL-II` dataset that we classify as `benign` turn out to all be benign NetBIOS clients that often made connection requests to absent hosts. The 3 benign **HH** hosts in the `ISP` dataset are all clients running P2P applications that attempt to contact a large number of transient peers that often do not respond. Most benign **LH** hosts are either low-profile NetBIOS clients (`LBL-II`) or P2P clients (`ISP`), and most benign **HL** hosts from both datasets are caused by Web clients accessing Web sites with many images stored elsewhere (e.g., a popular news site using Akamai's content distribution service, and a weather site having sponsor sites' images embedded).

Table 6.4 also shows that while those two thresholds are useful for nailing down a set of suspicious hosts (all in either **HH**, **LH**, or **HL**), a simple detection method based on fixed thresholds would cause 187 false positives because of benign hosts scattered in the **LH** and **HL** regions, as shown in Figure 6-8. However, using dynamic thresholds based on the observed behavior, RBS + TRW accurately identifies those 14 target hosts while significantly reducing false positives.

We evaluate RBS + TRW by varying λ_1 from λ_0 to $15\lambda_0$, and θ_1 from $\frac{1}{7}\theta_0$ to θ_0. We fix $\lambda_0 = 3.83$ Hz, $\theta_0 = 0.7$, $\beta = 0.99$, and $\alpha = 10^{-5}$. Figures 6-9 and 6-10 show the number

of detection and false positives for each pair of λ_1 and θ_1. In particular, for $\lambda_1 = \lambda_0$, the combined algorithm reduces to TRW (dashed vertical lines along the θ_1 axis), and when $\theta_1 = \theta_0$, to RBS (dashed vertical lines along the λ_0 axis).

Table 6.5: Evaluation of RBS + TRW vs. RBS and TRW: `LBL-II` has 9 scanners and `ISP` has 5 scanners. $\overline{N}|H_1$ represents the average number of first-contact connections originated from the detected hosts upon detection.

			`LBL-II`			`ISP`		
	λ_1	θ_1	False -	False +	$\overline{N}\|H_1$	False -	False +	$\overline{N}\|H_1$
RBS	$11\lambda_0$	$= \theta_0$	0	25	5.6	2	8	6.4
TRW	$= \lambda_0$	$\frac{1}{7}\theta_0$	7	5	18.5	0	3	10.0
RBS + TRW	$10\lambda_0$	$\frac{4}{7}\theta_0$	1	9	6.9	1	6	5.0

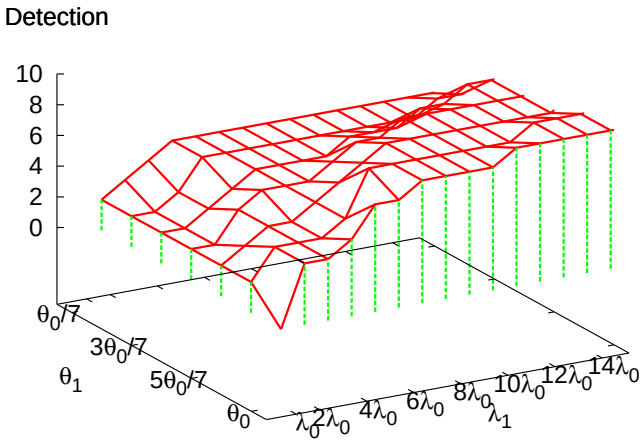
Table 6.5 compares the performance of the combined algorithm against that of RBS and TRW alone. First, we find the priors that make RBS (TRW) the most effective (0 false negatives) in identifying scanners in the `LBL-II` (`ISP`) dataset. The nature of our test datasets keeps either algorithm from working better across both datasets. In fact, when $\lambda_1 = 11\lambda_0$ and $\theta_1 = \theta_0$, RBS has 0 false negatives for `LBL-II`, but misses 2 **LH** scanners in `ISP`. In comparison, when $\lambda_1 = \lambda_0$ and $\theta_1 = \frac{1}{7}\theta_0$, TRW has 0 false negatives for `ISP`, but misses 7 scanners in `LBL-II`, including the **HL** hosts, 1 Web crawler and the **LH** nmap scanner.

We could address the problem of false negatives for either algorithm by running TRW and RBS in parallel, raising an alarm if either algorithm decides so. However, this approach comes at the cost of an increased number of false alarms, which usually result from **LH** hosts (e.g., Windows NetBIOS connections, often made to absent hosts) or **HL** hosts (e.g., a busy mail server or a Web proxy).

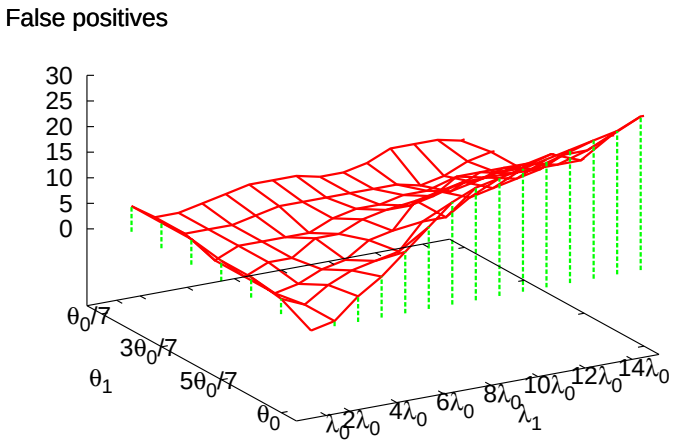
In general, improving the accuracy of a detection algorithm requires iterative adjustments of decision rules: first improving the detection rate by loosening the decision rule, and then decreasing the false positive rate by tightening the decision rule without losing too many correct detections. For this iteration, our combined algorithm, RBS + TRW provides two knobs, λ_1 and θ_1, that we can adjust to tune the detector to a site's traffic characteristics.

The trace-driven simulation shows that RBS + TRW with $\lambda_1 = 10\lambda_0$ and $\theta_1 = \frac{4}{7}\theta_0$ misses only two low-profile target hosts (one iprint host from `LBL-II` and a slow HTTP scanner from `ISP`) while generating no more than 15 false positives, per Table 6.5. Had we run RBS and TRW in parallel, we could have eliminated all the false negatives, but at the cost of 41 false alarms altogether.

Overall, RBS + TRW provides the good detection of high-profile worms and scanners (no more than 2 misses across both datasets) while generating less than 1 false alarm per hour for a wide range of parameters ($\lambda_1 \in [10\lambda_0, 15\lambda_0]$ and $\theta_1 \in [\frac{4}{7}\theta_0, \frac{6}{7}\theta_0]$), and reaching its detection decisions quickly (less than 7 first-contact connections on average).

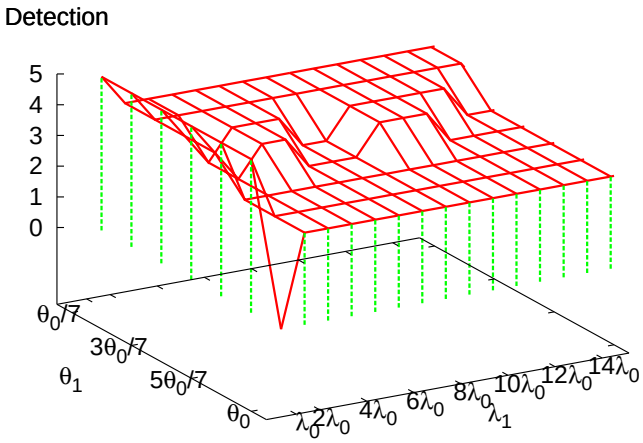


(a) Detection (out of 9 targets)

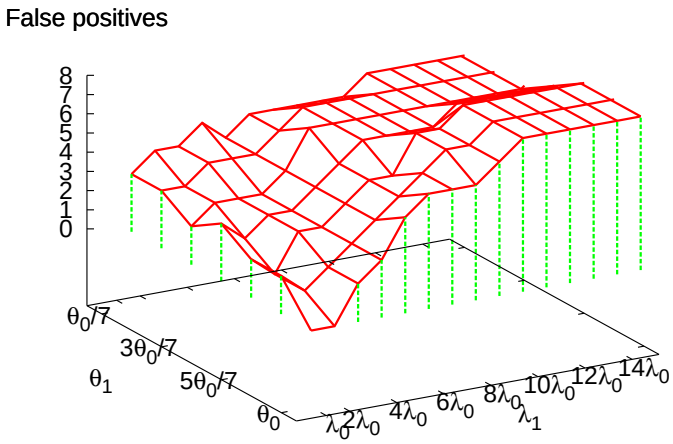


(b) False alarms (out of 4,008 hosts)

Figure 6-9: Simulation results of RBS + TRW for the `LBL-II` dataset, varying λ_1 and θ_1: each point represents the number of detection or false alarms when the detection algorithm is run with a certain pair of λ_1 and θ_1. In particular, points connected with the vertical lines along the θ_1 axis are the results when $\lambda_1 = \lambda_0$ (TRW) and those connected with the vertical lines along the λ_1 axis are the results when $\theta_1 = \theta_0$ (RBS).



(a) Detection (out of 5 targets)



(b) False alarms (out of 389 hosts)

Figure 6-10: Simulation results of RBS + TRW for the `ISP` dataset, varying λ_1 and θ_1: each point represents the number of detection or false alarms when the detection algorithm is run with a certain pair of λ_1 and θ_1. In particular, points connected with the vertical lines along the θ_1 axis are the results when $\lambda_1 = \lambda_0$ (TRW) and those connected with the vertical lines along the λ_1 axis are the results when $\theta_1 = \theta_0$ (RBS).

6.5 Discussion

This section discusses several technical issues that may arise when employing RBS + TRW in practice. While addressing these issues is beyond the scope of this work, we outline general ideas and directions.

Operational Issues. A worm detection device running RBS + TRW needs to maintain per local host information. For each host, a detector must track first-contact connections originated by the host, their failure/success status, and the elapsed time. The state thus increases proportional to the number of local hosts in the network (N) and the sum of all their currently pending first-contact connections. Given that RBS + TRW requires ≤ 10 first-contact connections on average to reach a decision (see Section 6.4), we can estimate amount of state as scaling on the order of $10N$. Note that every time RBS + TRW crosses either threshold, it resets its states for the corresponding host.

When constrained by computation and storage resources, one can employ cache data structures suggested by Weaver *et al.* [83] that track first-contact connections with a high precision. However, we note that running RBS + TRW on the aggregated traffic across hosts (as opposed to the per-host operation for which it is designed) can significantly affect the detection performance due to the uneven traffic distribution generated by each end-host [90].

Post-detection Response. The results in Table 6.5 correspond to RBS + TRW generating 0.07 false alarms per hour at the `LBL-II` site and 0.57 per hour at the `ISP` site. This low rate, coupled with RBS + TRW's fast detection speed, make it potentially suitable for automated containment, crucial to defending against fast-spreading worms. Alternatively, a network operator could employ connection rate-limiting for hosts detected by RBS + TRW, automatically restricting such hosts to a low fan-out rate.

Limitations. As indicated in Figure 6-8, RBS + TRW is unable to detect targeting worms using high-quality hit lists comprised of at least 70% active hosts and spreading no faster than several first-contact connections per second. Detecting such worms might be possible by working on larger time scales. For example, a scanner that generates first-contact connections at a rate of 1 Hz will end up accessing 3,600 different hosts in an hour, far outnumbering the sustained activity of a typical benign host. Thus, a natural avenue for future work is assessing the operation of RBS on longer timescales.

Finally, attackers can game our detection algorithm by tricking end users into generating first-contact connections either at a high rate (RBS), or that will likely end up failing (TRW). For instance, similar to an attack discussed in Section 5.4, an attacker could put content on a web site with numerous embedded links to non-existent destinations.

6.6 Summary

We have presented a worm detection algorithm, RBS (Rate-Based Sequential Hypothesis Testing), that rapidly identifies high-fan-out behavior by hosts (e.g., targeting worms) based on the rate at which the hosts initiate connections to new destinations. RBS is built on an empirically-driven probability model that captures benign network characteristics, which

allows us to discriminate between benign traffic and scanning traffic. Developed using the mathematical framework of sequential hypothesis testing [80], RBS can effectively *adapt* its threshold and window size, based on which it computes instantaneous rates to use in updating its decision-making in real-time. This gives RBS an essential advantage over schemes using single, fixed values for these.

We have evaluated RBS using trace-driven simulations. We find that when the factor of speed difference, n, between a scanner and a benign host is small, RBS requires more empirical data to arrive at a detection decision; for example, it requires on average 10.4 first-contact connections when $n = 5$, but the theoretical bound shows that it can detect any scanners that sustain more than 9.5 first-contact connections per second. In addition, as n grows larger RBS provides accurate and quick detection.

Finally, we have developed RBS + TRW, a hybrid of RBS and TRW, which combines *fan-out rate* and *probability of success* of each first-contact connection. RBS + TRW provides a unified framework for detecting fast-propagating worms independent of their target discovery strategy (i.e., targeting worm or scanning worm). Using two traces from two qualitatively different sites, containing 389 active hosts and 4,008 active hosts, respectively, we show that RBS + TRW provides fast detection of hosts infected by Code Red II, as well as the internal Web crawlers and a network monitoring tool that we use as proxies for targeting worms. In doing so, it generates less than 1 false alarm per hour.

Chapter 7

Conclusion and Future Directions

We have investigated three malicious network activities — portscan, scanning worm infection, and targeting worm propagation, and developed detection algorithms that accurately identify a host engaged in such forms of malicious activities in real-time. A common basis of our detection algorithms is an ability to capture connection-level characteristics that are inherent to each activity type. We have shown that there is a disparity between the frequency with which connections to newly visited addresses are successful for benign hosts vs. for scanners (Chapters 4 and 5). Another characteristic is the rate at which a host initiates connections to new destinations, which proves useful for identifying high-fan-out behavior by hosts infected by targeting worms (Chapter 6).

Our algorithms rely on connection-level information (e.g., source and destination IP addresses, protocol, connection status) and do not require access to packet contents. Compared to content-based approaches [37, 62, 81], our connection-based approach incurs significantly lower processing cost when implemented in a detection system. Moreover, payload information is not always available (e.g., failed connection attempts), and the increasing use of application-level encryption can render content-based approaches ineffective. However, we note that content-based approaches are capable of detecting slow-propagating (stealthy) worms that are indistinguishable from benign hosts in their traffic behavior.

Our study shows that due to the stochastic nature of a host's network behavior, it is important that a detection algorithm should be able to dynamically adapt its decision-making based on observed traffic data. Built on Wald's mathematical framework of sequential analysis [80], our detection algorithms automatically adjust the number of events to be collected according to the strength of the evidence. Furthermore, the sequential hypothesis testing methodology enables engineering the detection algorithms to meet false positive and false negative targets, rather than triggering when fixed thresholds are crossed.

While sequential analysis provides an underlying structure over which one can develop an adaptive real-time detection algorithm, the algorithm's performance greatly depends on how well *priors* model actual benign and malicious network activities. When poor priors are used, the performance bounds that the sequential analysis provides are no longer valid: false positives can occur because of atypical benign applications whose behavior is similar

to what is modeled as malicious. Likewise, false negatives can occur if the *a priori* models of benign activity include some cases of low-profile attack traffic.

In this study, we obtain priors based on the analysis of real network trace data that contain various types of network traffic as well as the malicious traffic of interest. However, we find it challenging to find "good" priors that properly separate benign network traffic from malicious mainly because of the following two reasons.

1. Once a detection algorithm is known, attackers can craft traffic behavior such that their attack traffic smears among other legitimate traffic flows. Because of this never-ending arms race between attackers and defense systems, it may be impossible to design a bullet-proof detection algorithm. Yet, some traffic properties are harder to modify and therefore work better as priors. In Chapter 4, we show that a strong modality appears when we look at the ratio of failed connection attempts over the total number of connections originated from each remote host. Since a portscanner has little knowledge of the configuration of a target network, this pattern of frequently accessing non-responding hosts (thus making a connection attempt unsuccessful) is hard to alter.

2. Applications vary a lot in their network behavior, making it difficult to find a reasonably simple model that captures all benign network activity. As shown in Chapter 6, a benign host's first-contact connection interarrival times are distributed across a wide range of intervals. However, when focusing on sub-second interarrivals, we find that those interarrival times fit an exponential distribution fairly well.

Using real trace data collected from multiple sites, we have evaluated the performance of our algorithms and examined false alarms and missed detections: our portscan detection algorithm, TRW, is more accurate and requires a much smaller number of events (4 or 5 packets in practice) to detect remote portscanners compared to previous schemes such as Bro [52] and Snort [56] (Chapter 4). Our credit-based rate limiting algorithm results in significantly fewer unnecessary rate limiting than the virus throttle [76] (Chapter 5). Our worm detection algorithm, RBS + TRW, provides fast detection of hosts infected by Code Red II, as well as hosts exhibiting abnormal level of high fan-out-rate. In doing so, RBS + TRW generates less than 1 false alarm per hour (Chapter 6).

This dissertation demonstrates that, when properly modeled, traffic characteristics can be used for identifying certain malicious network activities with high accuracy. One avenue for future work includes extending a detection algorithm to further classify the results with application-level semantics. For instance, if there is a database available that maintains information about which worms exploit which services, the detection algorithm can trigger an alarm of the "Slammer worm" as opposed to "some worm", if the detector finds a local host generating scan traffic to port 1434. Imbuing alarms with such semantics will facilitate the construction of an appropriate response after detection. If an alarm results from the traffic from the application in which a critical vulnerability is recently published, it is a strong indicator of a new worm outbreak and thus automatic containment is vital.

Finally, this dissertation provides important building blocks toward a general framework of categorizing network traffic by the activity being performed at the originating client. If traffic profiles are available for multiple activity types, one can build a classifier that analyzes network traffic associated with active hosts and reports their activity in real-time (e.g., worm propagation over port 80, running peer-to-peer applications tunneling through port 80, benign Web browsing). This kind of classifier will not only protect the network against many known Internet attacks, but also significantly enhance our understanding of ongoing "network situational awareness" [91].

Appendix A

Implementation of TRW in Bro policy

The Bro network intrusion detection system includes a number of policy scripts, which analyze network traffic and determine alarm worthy events used by the main program [1]. We implemented the Threshold Random Walk (TRW) portscan detection algorithm in Bro policy. One can load the TRW policy file, `trw.bro` using the @load command. Below is the source code of `trw.bro`.

```
@load alert
@load port-name
@load hot
@load scan

redef use_TRW_algorithm = T;

redef enum Alert += {
 TRWAddressScan, # Source flagged as scanner by TRW algorithm
 TRWScanSummary, # Summary of scanning activities reported by TRW
};

module TRW;

global target_detection_prob = 0.99 &redef;
global target_false_positive_prob = 0.01 &redef;

# As defined in Chapter 4, theta_zero (theta_one) is the success
# probability of a given benign (scanning) remote host's first
# contact connection
global theta_zero = 0.8 &redef;
global theta_one  = 0.1 &redef;

# Set of remote hosts successfully accessed by local hosts
global friendly_remotes: set[addr];

# Set of local honeypot hosts if any
```

```
global honeypot: set[addr];

# Upper and lower thresholds: initialized when Bro starts
global eta_zero: double;
global eta_one: double;

# Tell TRW not to flag friendly remotes
global do_not_flag_friendly_remotes = T &redef;

# Set of services of outbound connections that are
# possibly triggered by incoming connections
const triggered_outbound_services =
 { ident, finger, 20/tcp, } &redef;

global TRW_scan_sources: set[addr];
global TRW_benign_sources: set[addr];

global first_contact_connections: set[addr, addr];

global lambda: table[addr] of double &default = 1.0;
global num_scanned_locals: table[addr] of count &default = 0;

event bro_init() {
# Approximate solutions for upper and lower thresholds
 eta_zero =
  (1 - target_detection_prob) / (1 - target_false_positive_prob);
 eta_one = target_detection_prob / target_false_positive_prob;
 }

event TRW_scan_summary(orig: addr) {
 ALERT([$alert=TRWScanSummary, $src=orig,
  $msg=fmt("%s scanned a total of %d hosts",
  orig, num_scanned_locals[orig])]);
 }

export {
 function check_TRW_scan(c: connection, state: string,
 reverse: bool): bool {
  local id = c$id;
  local service =
   ( c$service == port_names[20/tcp] ) ? 20/tcp : id$resp_p;
  local orig = reverse ? id$resp_h : id$orig_h;
  local resp = reverse ? id$orig_h : id$resp_h;
  local outbound = is_local_addr(orig);

  # Mark a remote as friendly if it is successfully accessed by
  # a local with protocols other than triggered_outbound_services
```

```
if ( outbound ) {
 if ( resp !in TRW_scan_sources &&
  service !in triggered_outbound_services &&
  orig !in honeypot && state != "OTH" )
  add friendly_remotes[resp];

 return F;
}

if ( orig in TRW_scan_sources )
 return T;

if ( orig in TRW_benign_sources )
 return F;

if ( do_not_flag_friendly_remotes && orig in friendly_remotes )
 return F;

# Start TRW evaluation
local flag = +0;
local resp_byte = reverse ? c$orig$size : c$resp$size;
local established = T;

if ( state == "S0" || state == "REJ" || state == "OTH" ||
 ( state == "RSTOS0" && resp_byte <= 0 ) )
established = F;

if ( ! established || resp in honeypot ) {
 if ( [orig, resp] !in first_contact_connections ) {
  flag = 1;
  add first_contact_connections[orig, resp];
 }
}

else if ( [orig, resp] !in first_contact_connections ) {
 flag = -1;
 add first_contact_connections[orig, resp];
}

if ( flag == 0 )
 return F;

local ratio = 1.0;

# Update the corresponding likelihood ratio of orig
if ( theta_zero <= 0 || theta_zero >= 1 || theta_one <= 0 ||
 theta_one >= 1 || theta_one >= theta_zero ) {
```

```
    # Error: theta_zero should be between 0 and 1
    log "bad theta_zero/theta_one in check_TRW_scan";
    use_TRW_algorithm = F;
    return F;
   }

   if ( flag == 1 )
    ratio = (1 - theta_one) / (1 - theta_zero);

   if ( flag == -1 )
    ratio = theta_one / theta_zero;

   ++num_scanned_locals[orig];

   lambda[orig] = lambda[orig] * ratio;
   local updated_lambda = lambda[orig];

   if ( target_detection_prob <= 0 ||
    target_detection_prob >= 1 ||
    target_false_positive_prob <= 0 ||
    target_false_positive_prob >= 1 ) {
    # Error: target probabilities should be between 0 and 1
    log "bad target probabilities in check_TRW_scan";
    use_TRW_algorithm = F;
    return F;
   }

   if ( updated_lambda > eta_one ) {
    add TRW_scan_sources[orig];
    ALERT([$alert=TRWAddressScan, $src=orig,
     $msg=fmt("%s scanned a total of %d hosts",
     orig, num_scanned_locals[orig])]);
     schedule 1 day { TRW_scan_summary(orig) };
     return T;
   }

   if ( updated_lambda < eta_zero ) {
    add TRW_benign_sources[orig];
    return F;
   }
  }
}
```

Bibliography

[1] Bro Intrusion Detection System. http://bro-ids.org. (Cited on pages 78 and 111.)

[2] Internet Security Systems — Internet Scanner. http://www.iss.net/products_services/enterprise_protection/vulnerability_assessment/scanner_internet.php. (Cited on page 16.)

[3] Nessus. http://www.nessus.org/. (Cited on page 16.)

[4] Nmap — Free Security Scanner for Network Exploration & Security Audits. http://www.insecure.org/nmap/. (Cited on pages 16, 30 and 101.)

[5] Novell: iPrint Overview. http://www.novell.com/products/netware/printing/quicklook.html. (Cited on page 96.)

[6] WhatsUp Gold — The Standard for Network Monitoring Systems. http://www.areawidetech.com/whatsup_gold.htm. (Cited on page 95.)

[7] D. Anderson, T. Lunt, H. Javits, A. Tamaru, and A. Valdes. Detecting Unusual Program Behavior Using the Statistical Components of NIDES. Technical Report SRI-CSL-95-06, SRI Computer Science Laboratory, May 1995. (Cited on page 21.)

[8] Michéle Basseville and Igor V. Nikiforov. *Detection of Abrupt Changes: Theory and Application*, chapter 4.3: Sequential Analysis. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. (Cited on page 25.)

[9] Vincent Berk, George Bakos, and Robert Morris. Designing a Framework for Active Worm Detection on Global Networks. In *Proceedings of the IEEE International Workshop on Information Assurance*, March 2003. (Cited on pages 35 and 71.)

[10] Vincent Berk, Robert Gray, and George Bakos. Using Sensor Networks and Data Fusion for Early Detection of Active Worms. In *Proceedings of the SPIE Aerosense Conference*, April 2003. (Cited on pages 35 and 71.)

[11] CAIDA. CAIDA-Analysis of Code Red. http://www.caida.org/analysis/security/code-red/. (Cited on page 65.)

[12] CERT. CERT Advisory CA-2001-26 Nimda Worm. http://www.cert.org/advisories/CA-2001-26.html. (Cited on page 33.)

[13] CERT. "Code Red II:" Another Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. http://http://www.cert.org/incident_notes/IN-2001-09.html. (Cited on pages 34 and 79.)

[14] Shigang Chen and Yong Tang. Slowing Down Internet Worms. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, Tokyo, Japan, March 2004. (Cited on pages 35, 85 and 86.)

[15] CIO Asia. CA Details Sophisticated Web Attack. http://www.idg.com.sg/ShowPage.aspx?pagetype=2&articleid=1482&pubid=5&issueid=48. (Cited on page 15.)

[16] Fred Cohen. Computer Viruses: Theory and Experiments. *Computers and Security*, 6(1):22–35, 1987. (Cited on page 32.)

[17] Marco de Vivo, Eddy Carrasco, Germinal Isern, and Gabriela O. de Vivo. A Review of Port Scanning Techniques. *SIGCOMM Computer Communication Review*, 29(2):41–48, 1999. (Cited on page 30.)

[18] Sven Dietrich, Neil Long, and David Dittrich. Analyzing Distributed Denial Of Service Tools: The Shaft Case. In *Proceedings of LISA*, 2000. (Cited on page 17.)

[19] Holger Dreger, Anja Feldmann, Vern Paxson, and Robin Sommer. Operational Experiences with High-Volume Network Intrusion Detection. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, 2004. (Cited on page 20.)

[20] Mark Eichin and Jon Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1989. (Cited on pages 32 and 86.)

[21] F-Secure. F-Secure Virus Description Database. http://www.f-secure.com/v-descs/. (Cited on page 32.)

[22] F-Secure. F-Secure Virus Descriptions : Santy. http://www.f-secure.com/v-descs/santy_a.shtml. (Cited on page 86.)

[23] F-Secure Computer Virus Information page. LoveLetter. http://www.f-secure.com/v-descs/love.shtml. (Cited on page 33.)

[24] F-Secure Computer Virus Information page. Lovesan. http://www.f-secure.com/v-descs/msblast.shtml. (Cited on pages 34, 65 and 79.)

[25] F-Secure Computer Virus Information page. Mimail.J. http://www.f-secure.com/v-descs/mimail_j.shtml. (Cited on page 79.)

[26] F-Secure Computer Virus Information page. Mydoom. http://www.f-secure.com/v-descs/novarg.shtml. (Cited on pages 34 and 79.)

[27] F-Secure Computer Virus Information page. Nimda. http://www.f-secure.com/v-descs/nimda.shtml. (Cited on page 33.)

[28] F-Secure Computer Virus Information page. Zotob.A. http://www.f-secure.com/v-descs/zotob_a.shtml. (Cited on page 34.)

[29] Nick Feamster, Jaeyeon Jung, and Hari Balakrishnan. An Empirical Study of "Bogon" Route Advertisements. In *Computer Communication Review, Volume 35, Number 1*, January 2005. (Cited on page 65.)

[30] Simson Garfinkel, Gene Spafford, and Alan Schwartz. *Practical UNIX & Internet Security*, chapter 5.5: SUID. O'Reilly Media, Inc., Sebastopol, CA, 3rd edition, February 2003. (Cited on page 21.)

[31] Anup K. Ghosh, Aaron Schwartzbard, and Michael Schatz. Learning Program Behavior Profiles for Intrusion Detection. In *Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring*, April 1999. (Cited on page 21.)

[32] HackFix. SubSeven Removals. http://www.hackfix.org/subseven. (Cited on page 20.)

[33] L. Todd Heberlein, Gihan Dias, Karl Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A Network Security Monitor. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1990. (Cited on page 30.)

[34] Koral Ilgun, Richard A. Kemmerer, and Phillip A. Porras. State Transition Analysis: A Rule-Based Intrusion Detection Approach. *Software Engineering*, 21(3):181–199, 1995. (Cited on page 20.)

[35] Don Johnson. Criteria in Hypothesis Testing. http://cnx.rice.edu/content/m11228/latest/. (Cited on pages 25 and 26.)

[36] Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004. (Cited on page 22.)

[37] Hyang-Ah Kim and Brad Karp. Autograph: Toward Automated Distributed Worm Signature Detection. In *Proceedings of the 13th USENIX Security Symposium*, August 9–13, 2004. (Cited on pages 35 and 107.)

[38] C. Ko, M. Ruschitzka, and K. Levitt. Execution Monitoring of Security-Critical Programs in Distributed Systems: a Specification-Based Approach. In *Proceedings of the IEEE Symposium on Security and Privacy*, 1997. (Cited on page 21.)

[39] Sandeep Kumar and Eugene H. Spafford. A Pattern Matching Model for Misuse Intrusion Detection. In *Proceedings of the 17th National Computer Security Conference*, 1994. (Cited on page 20.)

[40] Christopher Leckie and Ramamohanarao Kotagiri. A Probabilistic Approach to Detecting Network Scans. In *Proceedings of the 8th IEEE Network Operations and Management Symposium (NOMS 2002)*, pages 359–372, Florence, Italy, April 2002. (Cited on page 31.)

[41] Ulf Lindqvist and Phillip A Porras. Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST). In *Proceedings of the 1999 IEEE Symposium on Security and Privacy*, pages 146–161, Oakland, California, may 1999. IEEE Computer Society Press, Los Alamitos, California. (Cited on page 20.)

[42] Matthew V. Mahoney and Philip K. Chan. Learning Nonstationary Models of Normal Network Traffic for Detecting Novel Attacks. In *Proceedings ACM SIGKDD international conference on Knowledge discovery and data mining*, NY, USA, 2002. (Cited on page 21.)

[43] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall PTR, 2004. (Cited on page 17.)

[44] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the Slammer worm. *IEEE Security and Privacy*, 1:33–39, July 2003. (Cited on pages 15, 29, 34 and 65.)

[45] David Moore and Colleen Shannon. The Spread of the Code-Red Worm (CRv2). http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml. (Cited on pages 33 and 65.)

[46] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage. Internet Quarantine: Requirements for Containing Self-Propagating Code. In *Proceedings of IEEE INFOCOM*, April 2003. (Cited on pages 34 and 66.)

[47] Robert Morris and Ken Thompson. Password Security: A Case History. *Communications of the ACM*, 22(11):594–597, 1979. (Cited on page 17.)

[48] George Moustakides. Optimal Procedures for Detecting Changes in Distributions. *Annals Statistics*, 14:1379–1387, 1986. (Cited on pages 22 and 35.)

[49] Peter G. Neumann. Risks of Passwords. *Communications of the ACM*, 37(4), 1994. (Cited on page 17.)

[50] Openwall Project. John The Ripper. http://www.openwall.com/john/. (Cited on page 17.)

[51] Martin Overton. Bots and Botnets: Risks, Issues and Prevention. In *Proceedings of the 15th Virus Bulletin Conference*, 2005. (Cited on page 17.)

[52] Vern Paxson. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999. (Cited on pages 20, 31, 78, 95 and 108.)

[53] J. B. Postel. *Transmission Control Protocol*, September 1981. RFC 793. (Cited on page 30.)

[54] Thomas H. Ptacek and Timothy N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998. (Cited on page 20.)

[55] Seth Robertson, Eric V. Siegel, Matt Miller, and Salvatore J. Stolfo. Surveillance Detection in High Bandwidth Environments. In *Proceedings of the 2003 DARPA DISCEX III Conference*, Washington, DC, April 2003. (Cited on pages 31 and 68.)

[56] Martin Roesch. Snort: Lightweight Intrusion Detection for Networks. In *Proceedings of the 13th Conference on Systems Administration (LISA-99)*, Berkeley, CA, November 1999. USENIX Association. (Cited on pages 20, 30 and 108.)

[57] Stuart E. Schechter, Jaeyeon Jung, and Arthur W. Berger. Fast Detection of Scanning Worm Infections. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, September 2004. (Cited on page 23.)

[58] R. Sekar and P. Uppuluri. Synthesizing Fast Intrusion Prevention/Detection Systems from High-Level Specifications. In *Proceedings of the 8th USENIX Security Symposium*, 1999. (Cited on page 21.)

[59] Colleen Shannon and David Moore. The Spread of the Witty Worm. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), March 2004. (Cited on pages 34 and 65.)

[60] Stelios Sidiroglou and Angelos D. Keromytis. A Network Worm Vaccine Architecture. In *Proceedings of the IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Workshop on Enterprise Security*, June 2003. (Cited on page 66.)

[61] Stelios Sidiroglou and Angelos D. Keromytis. Countering Network Worms Through Automatic Patch Generation. In *Proceedings of the IEEE Symposium on Security and Privacy*, 2005. (Cited on page 66.)

[62] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated Worm Fingerprinting. In *Proceedings of the 13th Operating Systems Design and Implementation OSDI*, December 2004. (Cited on pages 35 and 107.)

[63] Eugene H. Spafford. A Failure to Learn from the Past. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 217–233, December 2003. (Cited on pages 32 and 86.)

[64] @stake. L0phtCrack. http://www.atstake.com/products/lc/. (Cited on page 17.)

[65] Stuart Staniford. Containment of Scanning Worms in Enterprise Networks. *Journal of Computer Security*, Forthcoming. (Cited on page 35.)

[66] Stuart Staniford, James A. Hoagland, and Joseph M. McAlerney. Practical Automated Detection of Stealthy Portscans. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, Athens, Greece, 2000. (Cited on pages 32 and 58.)

[67] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to 0wn the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, Berkeley, CA, USA, August 2002. (Cited on page 86.)

[68] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS – A Graph-based Intrusion Detection System for Large Networks. In *Proceedings of the 19th National Information Systems Security Conference*, volume 1, pages 361–370, October 1996. (Cited on pages 35 and 84.)

[69] Symantec. Security Response – Code Red II. http://securityresponse.symantec.com/avcenter/venc/data/codered.ii.html. (Cited on pages 29, 34 and 79.)

[70] Symantec. Symantec Internet Security Threat Report Identifies More Attacks Now Targeting e-Commerce, Web Applications. http://www.symantec.com/press/cgi/printfriendlypress.cgi?release=2004/n040920b.html. (Cited on page 15.)

[71] Symantec Security Response. Search and Latest Virus Threats. http://www.symantec.com/avcenter/vinfodb.html. (Cited on page 32.)

[72] Symantec Security Response. W32.Nimda.A@mm. http://securityresponse.symantec.com/avcenter/venc/data/w32.nimda.a@mm.html. (Cited on page 33.)

[73] Symantec Security Response. W32.Sasser.Worm. http://securityresponse.symantec.com/avcenter/venc/data/w32.sasser.worm.html. (Cited on page 65.)

[74] silicon.com. Denial of Service Attack Victim Speaks Out. http://http://management.silicon.com/smedirector/0,39024679,39130810,00.htm, 2005. (Cited on page 17.)

[75] The Internet Assigned Numbers Authority. Port Numbers. http://http://www.iana.org/assignments/port-numbers/, last accessed on February 1, 2006. (Cited on page 79.)

[76] Jamie Twycross and Matthew M. Williamson. Implementing and Testing a Virus Throttle. In *Proceedings of the 12th USENIX Security Symposium*, August 2003. (Cited on pages 13, 35, 66, 75, 78, 80, 85, 88 and 108.)

[77] United States Computer Emergency Readiness Team. US-CERT Vulnerability Notes. http://www.kb.cert.org/vuls/. (Cited on page 17.)

[78] USA TODAY. Unprotected PCs can be Hijacked in Minutes. http://www.usatoday.com/money/industries/technology/2004-11-29-honeypot_x.htm. (Cited on page 15.)

[79] Luis von Ahn, Manuel Blum, and John Langford. Telling Humans and Computers Apart (Automatically) or How Lazy Cryptographers Do AI. Technical Report CMU-CS-02-117, February 2002. (Cited on page 80.)

[80] Abraham Wald. *Sequential Analysis*. J. Wiley & Sons, New York, 1947. (Cited on pages 18, 21, 26, 48, 50, 54, 92, 106 and 107.)

[81] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proceedings of the Eighth International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, September 2005. (Cited on page 107.)

[82] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A Taxonomy of Computer Worms. In *Proceedings of the ACM Workshop on Rapid Malcode*, Washington, DC, 2003. (Cited on pages 65, 86 and 99.)

[83] Nicholas Weaver, Stuart Staniford, and Vern Paxson. Very Fast Containment of Scanning Worms. In *Proceedings of the 13th USENIX Security Symposium*, August 2004. (Cited on pages 35, 83, 84, 86, 97 and 105.)

[84] David Whyte, Evangelos Kranakis, and P.C. van Oorschot. DNS-based Detection of Scanning Worms in an Enterprise Network. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'05)*, February 2005. (Cited on pages 35 and 86.)

[85] Wikipedia. Mydoom. http://en.wikipedia.org/wiki/MyDoom, last accessed on February 10, 2006. (Cited on pages 34 and 79.)

[86] Wikipedia. SQL Slammer Worm. http://en.wikipedia.org/wiki/SQL_slammer_worm, last accessed on February 10, 2006. (Cited on page 65.)

[87] Wikipedia. Timeline of Notable Computer Viruses and Worms. http://en.wikipedia.org/wiki/Notable_computer_viruses_and_worms, last accessed on February 10, 2006. (Cited on page 32.)

[88] Wikipedia. Zotob. http://en.wikipedia.org/wiki/Zotob, last accessed on February 10, 2006. (Cited on page 34.)

[89] Matthew M. Williamson. Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. In *Proceedings of The 18th Annual Computer Security Applications Conference (ACSAC 2002)*, December 2002. (Cited on pages 35, 66 and 75.)

[90] Cynthia Wong, Stan Bielski, Ahren Studer, and Chenxi Wang. Empirical Analysis of Rate Limiting Mechanisms. In *Proceedings of the Eighth International Symposium on Recent Advances in Intrusion Detection (RAID 2005)*, September 2005. (Cited on page 105.)

[91] Vinod Yegneswaran, Paul Barford, and Vern Paxson. Using Honeynets for Internet Situational Awareness. In *Proceedings of ACM Sigcomm Hotnets*, 2005. (Cited on page 109.)

[92] T. Ylonen and C. Lonvick Ed. *The Secure Shell (SSH) Protocol Architecture*. Internet Engineering Task Force, January 2006. RFC 4251. (Cited on page 17.)