Fast Portscan Detection Using Sequential Hypothesis Testing

Jaeyeon Jung, Vern Paxson, Arthur W. Berger, and Hari Balakrishnan

MIT Computer Science and Artificial Intelligence Laboratory
Cambridge, MA USA
{jyjung, awberger, hari}@csail.mit.edu

ICSI Center for Internet Research
and
Lawrence Berkeley National Laboratory
Berkeley, CA USA
vern@icir.org

Abstract

Attackers routinely perform random "portscans" of IP addresses to find vulnerable servers to compromise. Network Intrusion Detection Systems (NIDS) attempt to detect such behavior and flag these portscanners as malicious. An important need in such systems is prompt response*: the sooner a NIDS detects malice, the lower the resulting damage. At the same time, a NIDS should not falsely implicate benign remote hosts as malicious.*

Balancing the goals of promptness and accuracy in detecting malicious scanners is a delicate and difficult task. We develop a connection between this problem and the theory of sequential hypothesis testing *and show that one can model accesses to local IP addresses as a random walk on one of two stochastic processes, corresponding respectively to the access patterns of benign remote 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 remote host. We use this insight to develop* **TRW** *(Threshold Random Walk), an online detection algorithm that identifies malicious remote hosts. 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, while also providing theoretical bounds on the low (and configurable) probabilities of missed detection and false alarms. In summary, TRW performs significantly faster and also more accurately than other current solutions.*

1. Introduction

Many Internet attacks seen today begin with a *reconnaissance* phase in which the attacker probes a set of addresses at a site looking for vulnerable servers. In principle, this pattern of *port scanning* manifests quite differently from legitimate remote access to the site, and thus holds promise for providing a means by which a network intrusion detection system (NIDS) can identify an attacker at a stage early enough to allow for some form of protective response to mitigate or fully prevent damage.

A number of difficulties arise, however, when we attempt to formulate an effective algorithm for detecting port scanning. The first is that there is no crisp definition of the activity. For example, clearly an attempted HTTP connection to the site's main Web server is okay, while a sweep through the entire address space looking for HTTP servers is not okay (though see below). But what about connections to a few addresses, some of which succeed and some of which fail?

Another issue is the granularity of identity. Do we consider probes from adjacent remote addresses as part of a single reconnaissance activity? What about probes from merely nearby addresses, or disparate addresses which together form a clear "coverage" pattern? Similarly, the locality of the addresses to which the probes are directed might be quite tight (a set of ports at a single local address, or the same port across adjacent local addresses) or scattered about the site's address space.

There are temporal considerations as well as spatial ones. Over how much time do we track activity? Do we factor in the rate at which connections are made? 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.

A final 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 port scanning in order to find Web servers to index. In addition, some applications (e.g., SSH, some peer-to-peer and Windows applications, etc.) have modes in which they scan in a benign attempt to gather information or locate servers. Ideally, we would like to separate out such benign use from overtly malicious use. We would note, however, that the question of whether scanning by search engines is benign will ultimately be a *policy* decision that will reflect the site's view 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 is in terms of the amount of subsequent activity by the scanner: the activity itself can occur at a very slow rate, but we still want to *nip it in the bud*, i.e., detect it before it has gone very far; and, ideally, do so with few false positives. The algorithm we develop, Threshold Random Walk (TRW), can generally detect a scanner after 4 or 5 connection attempts, with high accuracy.

In our work we use traces from two sites to develop and assess our detection algorithm. Both sites operate the Bro NIDS, which has its own scan detection algorithm based on counting the number of different local addresses to which a remote address makes connections. The alerts from the NIDS thus give us a limited form of "ground truth." At one of the sites, these alerts are used to *block* subsequent activity from the scanner. Given the site's willingness to use this policy (which has been in place for a number of years), and from the site's operational follow-up of uncovering erroneous blocks of benign sources (primarily triggered by complaints received by the help desk), we argue that this blocking policy is known to have a low degree of false positives and a high degree of efficacy.[1]

Therefore, regarding the first issue above of how to define a scan, at a minimum we require that our algorithm perform well in detecting the activity flagged by Bro's algorithm. However, we would also like to detect lower-profile malicious probing, and we would like to show we can flag the same traffic that Bro does, but sooner. Speed of detection can be quite important for sites such as this one where the NIDS actively blocks scanners, because the sooner a scanner is detected, the less information it can glean before being blocked.

In general, we consider it acceptable to flag any remote host as a scanner if it turns out that that host would never subsequently make a useful connection, where "useful" means successfully established, data transferred, and, to the degree we can determine, the transfer was benign. Clearly, assessing this property requires an oracle, and so cannot be performed in real-time; but we can approximate it by a *what-if* assessment of our algorithm using trace-driven simulation. Also, it is important to note that we do not aim to detect *all* such scanners. We are content with detecting most of them, provided we do so quickly. In addition, since the data we have available from the two sites consists almost solely of TCP connection logs, we confine our analysis to detecting TCP scanners.

Regarding the other issues above, for simplicity we confine our notion of "identity" to single remote IP addresses. Thus, we do not aim to detect "distributed" scans [9]. We also focus on detecting scans of multiple local addresses, regardless of their location in the address space, and do not consider detecting "vertical" scans of a single host. In terms of time, we aim to detect scans that might be spread over intervals ranging up to hours (our traces are all 24 hours long), and we do not consider the particular rate at which a remote host attempts to make connections.

We pursue the problem in the general framework of *anomaly detection*, though unlike classical anomaly detection we model not only benign behavior but also malicious behavior. Initially, we had thought to develop an algorithm that would train on logs of past connections in terms of differences regarding to which portions of a site's address space, and to which services, benign and malicious hosts tend to attempt to connect, and then use those differences as priors for making Bayesian decisions. However, our analysis of the sites' connection logs revealed a sharp distinction between the activity of apparently benign hosts and malicious hosts simply in terms of the proportion of their connections that are successfully established, and so our final algorithm has the highly desirable properties that (1) it does not require training, and (2) it does not require reparameterization when applying it at different sites.

The development of the work is as follows. In §2, we discuss previous research on scan detection, and related work. In §3, we present the connection log data that motivates the general form of our detection algorithm. In §4, 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 §5, comparing it to that of other algorithms. We discuss issues for further work in §6, and summarize in §7.

1 The site reports that on the occasions when the blocking mechanism fails, it is generally a matter of a few hours before an attacker compromises a local host; while with the mechanism in place, it is a matter of days to weeks. Thus, thwarting the reconnaissance activity has great utility.

2. Related Work

As noted by Staniford *et al.*, there has been surprisingly little work on the problem of detecting scans [7]. Historically most scan detection has been 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) [2], which had rules to detect any source IP address connecting to more than 15 distinct destination IP addresses within a given time window. Such approaches have the drawback that once the window size is known it is easy for attackers to evade detection by simply increasing their scanning interval. Moreover, the algorithm can erroneously flag a legitimate access such as that of Web crawlers or proxies.

Snort [6] 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* [1]. 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 it suffers from the same drawbacks as NSM since both rely on the same metrics.

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 [4] 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.

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 [5]. 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 [3]. 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.

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 before been accessed. This can be particularly problematic for a sparsely populated network, where only small number of active hosts are accessed by benign hosts.

The final work on scan detection of which we are aware is that of Staniford *et al.* on SPICE [7]. 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 TRW.

3. 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 and ICSI. 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.

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

Table 1. Summary of datasets

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 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 6 in Table 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 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.[2] 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`.

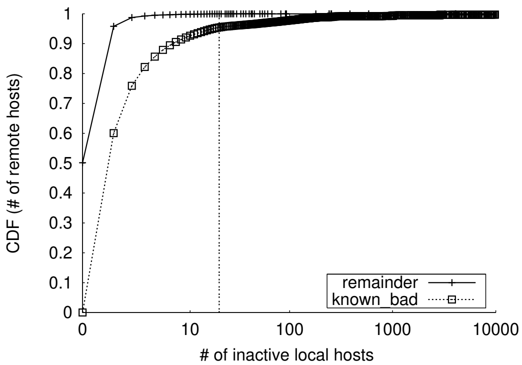
3.1. Separating Possible Scanners

As mentioned in the Introduction, 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 8 of Table 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.

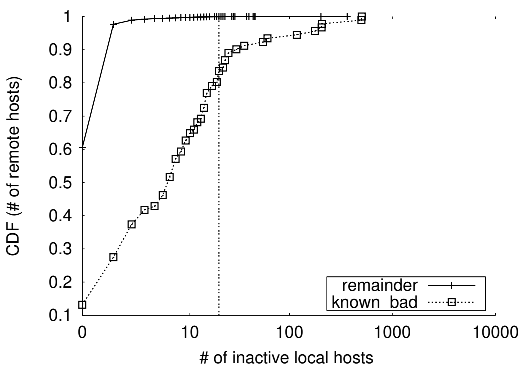
This is a difficult but crucial problem. We need to find a way to bootstrap our assessment of which of the remainder are likely, but undetected (due to their lower level of activity), scanners. Ideally, the means by which we do so would be wholly separate from our subsequently developed detection algorithm, 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 hosts 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 basis for distinguishing undiagnosed scanners from benign hosts, constructing a more accurate ground truth.

2 The firewall configurations at the two sites differ, but for brevity we omit the details.

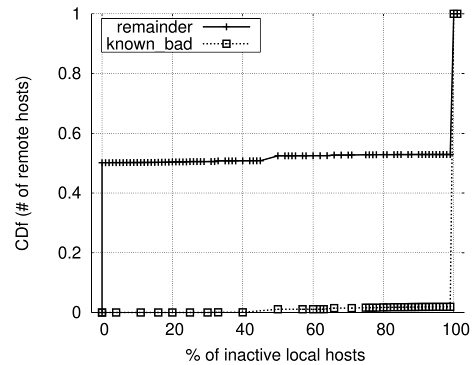


(a) LBL

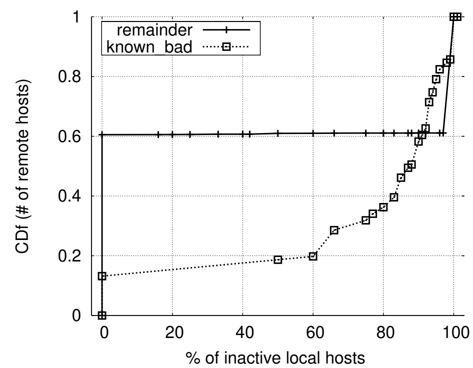


(b) ICSI

Figure 1. 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 hosts). The vertical dotted line at $x = 20$ corresponds to the threshold used by the Bro NIDS at the sites.



(a) LBL



(b) ICSI

Figure 2. 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 (% of inactive local hosts)

Figure 1 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 basis 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[3] 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 1 (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. Define `inactive_pct` as the percentage of the local hosts that a given remote host has accessed for which the connection attempt failed (was rejected or unanswered). Figure 2 shows the distribution of `inactive_pct` for each dataset. Figure 2(a) shows that about 99.5% of LBL's `known_bad` remotes hit nothing but inactive hosts (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% `inactive_pct`, or 100% `inactive_pct`—which

3 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.

			LBL	ICSI
Total unique remote hosts			190,928	29,528
known_bad			74,542	91
	scanners		122	7
	HTTP worms		37	69
remainder			116,386	29,437
	benign		61,456	17,974
		HTTP	47,343	6,026
	suspect		54,921	11,463
		HTTP	40,413	11,143

Table 2. Remote Host Characteristics: $<$ 80% inactive_pct is used to separate benign hosts from possible scanners.

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% inactive_pct are potentially benign,[4] while hosts with higher values of inactive_pct will be treated as possible scanners. We term these latter as suspect. Table 2 summarizes the resulting classifications, and also the proportion due to remote hosts accessing HTTP, since those dominate the *remainder*.

Finally, if we repeat the plots in Figure 1 for suspect hosts, we find that they exhibit distributions quite similar to those for known_bad hosts, which provides additional supporting evidence that the simple inactive_pct criteria works well for differentiating between benign hosts and scanners.

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

4.1. 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 [8].

4.2. Sequential Hypothesis Testing

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:

$$\begin{array}{ll} \Pr[Y_i = 0|H_0] = \theta_0, & \Pr[Y_i = 1|H_0] = 1 - \theta_0 \\ \Pr[Y_i = 0|H_1] = \theta_1, & \Pr[Y_i = 1|H_1] = 1 - \theta_1 \end{array} \quad (1)$$

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

$$\theta_0 > \theta_1.$$

Given the 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*.

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:

$$P_F \le \alpha \;\text{ and }\; P_D \ge \beta \quad (2)$$

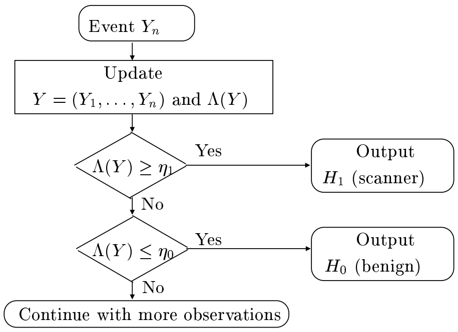


Figure 3. Flow diagram of the real-time detection algorithm

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

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). Following [8], as each event is observed we calculate the likelihood ratio:

$$\Lambda(Y) \equiv \frac{\Pr[Y|H_1]}{\Pr[Y|H_0]} = \Pi_{i=1}^{n} \frac{\Pr[Y_i|H_1]}{\Pr[Y_i|H_0]} \tag{3}$$

where Y is the vector of events observed so far and $\Pr[Y|H_i]$ represents the conditional probability mass function of the event stream Y given that model H_i is true; and where the second equality in (3) follows from the i.i.d. assumption. The likelihood ratio is then compared to an *upper* threshold, η_1, and a *lower* threshold, η_0. If $\Lambda(Y) \leq \eta_0$ then we accept hypothesis H_0. If $\Lambda(Y) \geq \eta_1$ then we accept hypothesis H_1. If $\eta_0 < \Lambda(Y) < \eta_1$ then we wait for the next observation and update $\Lambda(Y)$. See Figure 3.

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.[5]

The thresholds η_1 and η_0 should be chosen such that the false alarm and detection probability conditions, (2) 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, and they are *independent* of (1), the Bernoulli distributions conditioned on hypotheses H_0 and H_1. While these distributions 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.

To develop the previous point, 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 $Y_1, Y_2, \ldots, Y_n$, where on the n^{th} observation the upper threshold η_1 is hit and hypothesis H_1 is selected. Thus:

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

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:

$$\eta_1 \leq \frac{P_D}{P_F} \tag{4}$$

Analogous reasoning yields a lower bound for η_0:

$$\frac{1 - P_D}{1 - P_F} \leq \eta_0 \tag{5}$$

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.

$$\eta_1 \leftarrow \frac{\beta}{\alpha} \qquad \eta_0 \leftarrow \frac{1-\beta}{1-\alpha} \tag{6}$$

Since we derived the bounds (4) and (5) for arbitrary values of the thresholds, these bounds of course apply for this particular choice. Thus:

$$\frac{\beta}{\alpha} \leq \frac{P_D}{P_F} \qquad \frac{1 - P_D}{1 - P_F} \leq \frac{1-\beta}{1-\alpha} \tag{7}$$

Taking the reciprocal in the first inequality in (7) and noting that since P_D is between zero and one, $P_F < P_F/P_D$, yields the more interpretively convenient expression:

$$P_F < \frac{\alpha}{\beta} \equiv \frac{1}{\eta_1} \tag{8}$$

Likewise, for the second inequality in (7), noting that $1 - P_D < (1 - P_D)/(1 - P_F)$ yields:

$$1 - P_D < \frac{1-\beta}{1-\alpha} \equiv \eta_0 \tag{9}$$

5 To be more precise, it is a random walk in the logarithm of the likelihood-ratio space.

Equation (8) says that with the chosen thresholds (6), 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, equation (9) 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. Lastly, cross-multiplying in the two inequalities in (7) and adding yields:

$$1 - P_D + P_F \leq 1 - \beta + \alpha. \tag{10}$$

Equation (10) says 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 (6) to determine the thresholds η_0 and η_1, with resulting inequalities (8) - (10). An alternative viewpoint is that the user directly chooses the thresholds, η_0 and η_1, with knowledge of the inequalities (8) and (9). In summary, setting the thresholds to the approximate values of (6) is simple, convenient, and within the realistic accuracy of the model.

4.3. Number of Observations to Select Hypothesis

Given the performance criteria, (2), and the associated thresholds, (6), 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 [8], we present approximate expressions for the expected value of N (see Appendix I for a discussion of the tail probability of N).

For the analysis of N, it is convenient to consider the log of the likelihood ratio, (3), and view the resulting expression as a random walk:

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

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 $S_0 = 0$.)

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. Appendix I provides expressions for $E[S_N]$ and $E[X_i]$. Combining (6), (12), and (13), we obtain the approximate result:

$$\begin{aligned} E[N|H_0] &= \frac{\alpha \ln \frac{\beta}{\alpha} + (1-\alpha)\ln\frac{1-\beta}{1-\alpha}}{\theta_0 \ln \frac{\theta_1}{\theta_0} + (1-\theta_0)\ln\frac{1-\theta_1}{1-\theta_0}}, \\ E[N|H_1] &= \frac{\beta \ln \frac{\beta}{\alpha} + (1-\beta)\ln\frac{1-\beta}{1-\alpha}}{\theta_1 \ln \frac{\theta_1}{\theta_0} + (1-\theta_1)\ln\frac{1-\theta_1}{1-\theta_0}}. \end{aligned} \tag{11}$$

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

As shown in Equation (11), $E[N|H_0]$ 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.

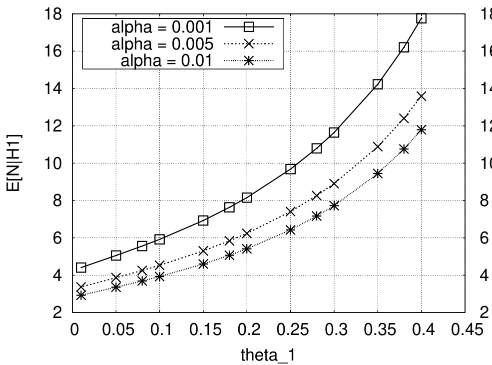
Assuming a scanner picks IP addresses at random, θ_1—the probability that it chooses an IP address with the requested service on—depends on the density of these addresses in a monitored network. Figure 4(a) shows how $E[N|H_1]$ changes as θ_1 increases. With $\alpha = 0.01$, $\beta = 0.99$, and $\theta_0 = 0.8$, $E[N|H_1]$ 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 §3.) In general, it takes longer to tell one model from the other the closer the two models are to each other. Figure 4(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 hosts, 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 hosts 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.

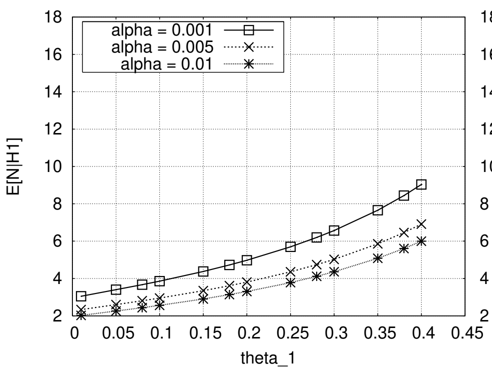
In those circumstances where such problematic hosts can be controlled, however, then we can configure the detection algorithm to use a higher θ_0, and thus enable it to make a faster decision. Figure 4(b) shows $E[N|H_1]$ when θ_0 is set to 0.9. The decrease in detection time is significant.

4.5. 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.



(a) $\theta_0 = 0.8$



(b) $\theta_0 = 0.9$

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

The bounds for upper and lower thresholds (Equation(4) and (5)) 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 [8]. 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. 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(3). Instead, the likelihood ratio should be calculated using a joint probability distribution, which complicates the computation.

5. 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.

5.1. Trace-driven Simulation

We use the datasets described in §3 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`.[6]
3) Check whether d already belongs to D_s. If so, skip the next two steps and proceed to the next line.
4) Update D_s with d, and update the likelihood ratio, L_s using Equation(3).
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 3 shows the simulation results for LBL and ICSI datasets. We excluded remote hosts that accessed less than

6 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.

Type		LBL				ICSI			
		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	13257	-	-	-	0	-	-	-
	H_1	13059	0.985	4.0	10	0	-	-	-
	H_0	15	-	5.1	10	0	-	-	-
	`PENDING`	183	-	-	11	0	-	-	-
`benign`	Total	2811	-	-	-	96	-	-	-
	H_1	33	-	8.1	24	0	-	-	-
	H_0	2343	-	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

Table 3. Simulation results when $P_D = 0.99$, $P_F = 0.01$, $\theta_1 = 0.2$, and $\theta_0 = 0.8$

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%)

Table 4. Break-down of "`suspects`" flagged as H_1.

4 distinct IP addresses from this table because with $P_D = 0.99$, $P_F = 0.01$, $\theta_1 = 0.2$, and $\theta_0 = 0.8$, TRW requires at least 4 observations to make a decision. The results depend on the parameter values; we present here results based on typical settings, where the detection probability should be at least 0.99 and the false alarm rate no larger than 0.01. We chose θ_1 and θ_0 based on the discussion in §3. Although we found 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.

First, we group remote hosts into the categories defined in §3 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 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 (generally perceived as hostile, but sometimes subject to "misfire"[7]).

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.

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

7 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.

		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

Table 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 3.

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 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 failed connections if they've previously received outbound connections from the site, we were able to identify all of the `suspect` remote hosts. Over 80% made nothing but failed `HTTP` connections, and we therefore suspect them as undetected worms. Table 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 $\theta_1 = 0.2$, the approximate solution for $E[N|H_1]$ is 5.4 as shown in §4.4, and our trace-driven simulations are consistent with this figure. This suggests that the parameters chosen for θ_0 and θ_1 closely 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(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.

Finally, to quantify the effectiveness of TRW, we use the two measures proposed by Staniford *et al.* [7]:

- *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 5 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.

5.2. 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 paper, 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 ad-

Type	Total	TRW H_1	TRW $\overline{N}$	TRW Max N	Bro H_1	Bro $\overline{N}$	Bro Max N	Snort H_1	Snort $\overline{N}$	Snort 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 6. Comparison of the number of H_1 across three categories for LBL dataset

Type	Total	TRW H_1	TRW $\overline{N}$	TRW Max N	Bro H_1	Bro $\overline{N}$	Bro Max N	Snort H_1	Snort $\overline{N}$	Snort 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

Table 7. Comparison of the number of H_1 across three categories for ICSI dataset

dresses/minute. Tables 6 and 7 show the number of (non-local) hosts reported as H_1 by the three algorithms.

Table 8 compares the efficiency and effectiveness across the three algorithms for both datasets. Note that two measures for TRW differ from Table 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 provide fast detection. But, as shown in Tables 6 and 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.[8] 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.

6. Discussion and Future Work

In this section we look at a number of additional dimensions to the problem space. Addressing these is beyond the scope of the present work, but we sketch our thinking on how we will pursue them in our future work.

Leveraging Additional Information. TRW's performance is somewhat remarkable given the limited information it uses. Potential refinements include: (1) factoring in

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

Table 8. Comparison of the efficiency and effectiveness across TRW, Bro, and Snort

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; (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 (per our original plan for anomaly detection outlined in the Introduction).

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, imag-

8 See §2 for the definitions of Y and Z

ine 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 §5, 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 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).

7. 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.

8. Acknowledgements

The authors would like to thank Magdalena Balazinska, Nick Feamster, Stuart Schechter, Robin Sommer, and Stuart Staniford for their comments on earlier drafts of this paper, and our shepherd, John McHugh.

References

[1] Nmap — free security scanner for network exploration & security audits. `http://www.insecure.org/nmap/`.

[2] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. In *Proc. IEEE Symposium on Research in Security and Privacy*, pages 296–304, 1990.

[3] C. Leckie and R. Kotagiri. A probabilistic approach to detecting network scans. In *Proceedings of the Eighth IEEE Network Operations and Management Symposium (NOMS 2002)*, pages 359–372, Florence, Italy, Apr. 2002.

[4] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks (Amsterdam, Netherlands: 1999)*, 31(23–24):2435–2463, 1999.

[5] S. Robertson, E. V. Siegel, M. Miller, and S. J. Stolfo. Surveillance detection in high bandwidth environments. In *Proceedings of the 2003 DARPA DISCEX III Conference*, pages 130 – 139, Washington, DC, 2003. IEEE Press. 22-24 April 2003.

[6] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 13th Conference on Systems Administration (LISA-99)*, pages 229–238, Berkeley, CA, Nov. 7–12 1999. USENIX Association.

[7] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. In *Proceedings of the 7th ACM Conference on Computer and Communications Security*, Athens, Greece, 2000.

[8] A. Wald. *Sequential Analysis*. J. Wiley & Sons, New York, 1947.

[9] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: global characteristics and prevalence. In *Proceedings of the 2003 ACM SIGMETRICS*, volume 31, 1 of *Performance Evaluation Review*, pages 138–147, New York, June 11–14 2003. ACM Press.

Appendix I
Conditional Expectation and Tail probability of N

Following Wald, [8], we provide expressions for the conditional expectation of S_N and X_i whose ratio is the conditional expectation of N, $E[N|H_j]$, $j = 0, 1$. 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,

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

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

$$\begin{aligned} E[X_i|H_0] &= (1-\theta_0)\ln\frac{1-\theta_1}{1-\theta_0} + \theta_0 \ln\frac{\theta_1}{\theta_0} \\ E[X_i|H_1] &= (1-\theta_1)\ln\frac{1-\theta_1}{1-\theta_0} + \theta_1 \ln\frac{\theta_1}{\theta_0} \end{aligned} \quad (12)$$

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

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

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

$$\begin{aligned} E[S_N|H_0] &= \alpha \ln \eta_1 + (1-\alpha)\ln \eta_0 \\ E[S_N|H_1] &= \beta \ln \eta_1 + (1-\beta)\ln \eta_0 \end{aligned} \quad (13)$$

Combining (6), (12), and (13), we obtain the approximate result in Equation(11).

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.*$N \le n_o$. Conditioning on the hypothesis for which hitting the upper threshold is more likely, H_1, we have:

$$\Pr[\sum_{i=1}^{n_o} X_i \ge \ln \eta_1 | H_1] \quad \le \quad \Pr[N \le n_o | H_1] \quad (14)$$

Normalizing the left hand side of (14) to mean zero variance one, yields:

$$\Pr\left[\frac{\sum_{i=1}^{n_o} X_i \; - \; n_o \mathrm{E}[X_i|H_1]}{\sqrt{n_o}\cdot\sigma(X_i|H_1)} \;\ge\; \frac{\ln \eta_1 \; - \; n_o \mathrm{E}[X_i|H_1]}{\sqrt{n_o}\cdot\sigma(X_i|H_1)} | H_1\right] \quad (15)$$

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 (15) 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 $N \le n_o$). Thus,

$$\Pr[N \le n_o|H_1] \;\approx\; 1 - \Phi\left(\frac{\ln \eta_1 \; - \; n_o\mathrm{E}[X_i|H_1]}{\sqrt{n_o}\cdot\sigma(X_i|H_1)}\right) \quad (16)$$

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

Analogous reasoning for the lower threshold and conditioning on H_0 yields

$$\Pr[N \le n_o|H_0] \;\approx\; \Phi\left(\frac{\ln \eta_0 \; - \; n_o\mathrm{E}[X_i|H_0]}{\sqrt{n_o}\cdot\sigma(X_i|H_0)}\right) \quad (17)$$