Network Scan Detection with LQS: A Lightweight, Quick and Stateful Algorithm

Mansour Alsaleh, P.C. van Oorschot
School of Computer Science
Carleton University, Ottawa, Canada
{malsaleh, paulv}@scs.carleton.ca

ABSTRACT

Network scanning reveals valuable information of accessible hosts over the Internet and their offered network services, which allows significant narrowing of potential targets to attack. Addressing and balancing a set of sometimes competing desirable properties is required to make network scanning detection more appealing in practice: 1) fast detection of scanning activity to enable prompt response by intrusion detection and prevention systems; 2) acceptable rate of false alarms, keeping in mind that false alarms may lead to legitimate traffic being penalized; 3) high detection rate with the ability to detect stealthy scanners; 4) efficient use of monitoring system resources; and 5) immunity to evasion. In this paper, we present a scanning detection algorithm designed to accommodate all of these goals. *LQS* is a fast, accurate, and light-weight scan detection algorithm that leverages the key properties of the monitored network environment as variables that affect how the scanning detection algorithm operates. We also present what is, to our knowledge, the first automated way to estimate a reference baseline in the absence of ground truth, for use as an evaluation methodology for scan detection. Using network traces from two sites, we evaluate LQS and compare its scan detection results with those obtained by the state-of-the-art TRW algorithm. Our empirical analysis shows significant improvements over TRW in all of these properties.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; C.2.3 [**Network Operations**]: Network monitoring

General Terms

Security, Algorithms

Keywords

Scanning Detection, Port Scanning, Host Discovery Techniques, Reconnaissance

1. INTRODUCTION

Network scanning continues to be a common reconnaissance technique that precedes many of today's Internet attacks. Many botnets and network worms scan various IP address ranges to locate vulnerable machines to attack [14, 15, 24]. Scanning is also an effective way to search for potential weaknesses in dedicated servers since pull-based infection techniques (e.g., drive-by downloads) and other infection techniques that require user interaction (e.g., opening malicious email attachments) are not applicable.

A single scan activity attempts to connect to a specific port in a host either to find out if the host is active or if the port is open and what service it offers. Given that the objective of network scanning is to find responsive services, scanners cannot avoid making failed connection attempts. Therefore, detection approaches based on a remote's failed connection attempts offer more promise where other detection features can be evaded by informed adversaries.

Most post-detection responses (e.g., limiting the amount of information that a scanner can learn about the monitored network by blocking some of their inbound network traffic) require fast, real-time detection of scanners. The fewer failed connection attempts by a remote host required by a detection algorithm to flag the remote as a scanner, the faster the scan detection and the more stealthy scanners are detected. In addition to the challenge of selecting an appropriate trade-off between the false positive rate and the number of required failed connection attempts, it is also important to balance between efficient use of monitoring system resources and reasonable accuracy of the detection algorithm.

In this paper, we propose a lightweight quick and stateful (*LQS*) real-time network scanning detection algorithm for external scanners. LQS leverages key properties of the operating environment that impact the detection performance such that they are incorporated into operational parameters of the algorithm. Our analysis and empirical evaluation finds that while LQS requires a small memory footprint to operate, its detection accuracy and speed outperforms the TRW algorithm [9]. Unlike TRW, LQS can detect vertical scans and it has a greater immunity to evasion from scanners who have a priori knowledge of some available services in the target network.

Contributions. Our main contributions are the following:

1. Lightweight Quick and Stateful Online Scan Detection Algorithm: We propose a lightweight network scan detection algorithm (LQS) that detects scanners as early as from their second connection at-

tempt to the monitored network. Unlike previous scan detection approaches (e.g., [9, 19]), LQS keeps the state of offered network services over time to evaluate inbound connection attempts.

2. Empirical Evaluation: We evaluate the performance of LQS on two datasets from two qualitatively different network environments and compare its results to those obtained by TRW.

3. Scan Detection Evaluation Methodology: We present an evaluation methodology for scan detection schemes in which remote hosts contacting the monitored network are classified after monitoring their network traffic over a relatively long period of time (as opposed to a short monitoring window in real-time scan detection to make a fast decision, as in the LQS algorithm). The new methodology provides a reference baseline for evaluation for each dataset studied, in the absence of ground truth.

Our implementation of LQS (Section 3.2, Algorithm 1) as a policy in the Bro IDS [1] is available at `http://lqs-bro.sourceforge.net/`. Our empirical evaluation shows that LQS both detects scanners earlier than TRW and has higher detection accuracy (e.g., in one dataset, LQS detection rate is 76% vs. 12% in TRW).

Organization. We discuss identification of scanners in Section 2. Section 2.1 describes the datasets used and their network environment. Section 2.2 presents a new methodology to obtain a reference baseline for evaluating network scanning detectors. Challenges in real-time scan detection are discussed in Section 2.3. We present a design overview of LQS in Section 3. Section 4 explores the advantages of LQS relative to TRW discussing the features and capabilities of both. Section 5 evaluates LQS on two datasets from different sites; scan detection results of both the LQS and TRW algorithms are given and analyzed. Section 6 discusses related work. Section 7 concludes.

2. IDENTIFICATION OF SCANNERS: ANALYSIS OF SCANNING PATTERNS

Typically, network scanners tend to probe a range of network addresses in search of active services of particular interest to the scanners. Unlike legitimate network traffic, most scanners' connection attempts are expected to fail since the density of network services (i.e., the ratio of open ports to closed ports of all Internet-addressable local hosts) in a given network is very small. Using failed connection attempts as a sign of scanning intent seems effective, as scanners cannot evade probing non-existing network services.

To use a network service remotely, the common way for a regular user to locate the IP address of the server in question is through DNS requests. Users usually enter the human-readable host name of the required server in the used application (e.g., entering a URL in a browser) which in turn sends a DNS request to obtain the corresponding IP address. The application often determines the appropriate destination port to contact the corresponding server. While it may seem unlikely for a benign remote host to make unsuccessful connections, in practice, there are several inevitable benign reasons to generate failed connection attempts (e.g., network failures, outdated DNS entries, and temporarily unavailable network services).

In Section 2.2, we study failed connection attempts in two datasets and propose a scan detection evaluation methodology. An overview of these datasets is first given in Section 2.1 below. Section 2.3 discusses challenges in real-time scan detection.

2.1 Overview of Datasets

Dataset I. The dataset is a full capture network trace collected at a class C university network with 62 Internet-addressable IP addresses. The trace was gathered over the period of Jan 28 to Mar 13, 2007 (45 days). The size of the dataset is 41 gigabytes. The active IP addresses during the capture period were 30. The network firewall allows inbound connection attempts to closed ports and unassigned IP addresses. Since local hosts respond to inbound connection attempts that are sent to closed ports, most inbound timed-out TCP connection attempts are destined to unassigned IP addresses (or turned off machines). Note that about 95% of inbound TCP connections in Table 1 are rejected (i.e., RST packet is sent by the destination) suggesting a high-volume of network scanning traffic. Few IP addresses used P2P file sharing over short bursts of the log capture period. Connection attempts to unavailable peers contributed to the rejected and timed-out outbound TCP connections.

To identify the network protocols running in the open ports in this network without relying on the port number, we used a signature-based detection method based on Ethereal display filter reference [2]. Six open ports (in three dedicated servers) were identified running the following network protocols: HTTP, HTTPS, SSH, SMTP, IMAPS, and IPP.

Dataset II. This is a network trace of packet headers collected at a class C university network (a network different than that of dataset I) with 254 Internet-addressable IP addresses. The trace was gathered over the period of Jun 17 to Jul 4, 2010 (18 days). The size of the dataset is 188 gigabytes. The active IP addresses during the capture period were 223. Inbound connection attempts to closed ports or unassigned IP addresses were not allowed by the network firewall. About 70% of inbound TCP connections are timed-out (i.e., did not go through the firewall) suggesting a high-volume of network scanning traffic.

Network protocols running in the open ports were identified by the same signature-based method used in the first dataset. Only protocol signatures located in the first bytes of the TCP payload data for packets with shorter than maximum header size are identified. The open ports fall into the following categories: (a) 180 ports running Sophos antivirus remote management system (port 8194); (b) 170 ports running Microsoft Directory Service (Microsoft-DS; e.g., SMB protocol); (c) 12 ports running Line Printer Daemon protocol (LPD; port 515); (d) 10 ports running Telnet protocol; (e) 7 ports running SSH protocol; and (f) 72 various other services mostly on ephemeral ports. There were no P2P protocols observed during the capture period.

2.2 Evaluation in the Absence of Ground Truth

A labeled dataset is often used to validate an intrusion detection technique. Accurate labeling of a dataset requires either unique signatures to match against or artificially created or injected intrusion traffic. A network scanning event

Number of:	Dataset I		Dataset II	
	Inbound	Outbound	Inbound	Outbound
a) Flows (TCP, UDP, and ICMP)	4,011,132	828,988	660,877	27,868,693
b) TCP connections (flows)	3,857,660	719,273	207,988	22,747,160
i) Successful TCP connections (percentage of b)	4.2%	57.2%	29.2%	71.7%
ii) Rejected TCP connections	95.79%	12.9%	2.2%	20.2%
iii) Timed-out TCP connections	0.01%	29.9%	68.6%	8.1%
c) Source IP addresses initiating TCP connections	7,031	30	28,922	223

Table 1: Datasets statistics (dataset I of Jan 28 to Mar 13, 2007; dataset II of Jun 17 to Jul 4, 2010)

could resemble legitimate traffic depending on (unknowable) intent, and thus general signatures for all network scanning events do not seem possible. Given the difficulty of generating synthetic traffic that represents all forms of network scanning, and that is distinguishable from legitimate traffic, simulation and emulation approaches that involve generating scanning events appear challenging for validation. Alternatively, aggregate behaviour of multiple events (e.g., frequency, rate, and the number of distinct destination IP addresses the remote made failed connection attempts to) from the same source can be used to infer scanning intent and to provide a reference baseline, that while not representing a solid *ground truth* of scanners, may give a *limited form* or *an estimated* ground truth.

Unlike real-time scan detection algorithms, which are typically designed for fast detection upon observing as few as possible connection attempts from remote hosts, the full network traffic of remote hosts (of a particular dataset) is available to establish a reference baseline of scanners. Although monitoring network traffic over a relatively long period of time (e.g., few days) gives us more confidence in identifying scanners, those with few connection attempts remain hard to identify.

Given the possible change of state in a remote host from benign to scanner and vice versa, the aggregate behaviour of the remote host over a relatively long period of time may seem inaccurate. Thus, it is important to consider the time parameter in which the remote is classified as a scanner for some time periods and benign for others. However, in a given remote host, the probability that both a scanning malware (e.g., a worm) and a legitimate software (e.g., browsing a Web site) contact the same network is low. Therefore, considering the change of state is not necessary in such a classification.

For the two datasets in Section 2.1, we attempt to generate a reference baseline (RB) for each remote host based on the following metrics:

1. the number of distinct {local IP address, destination port} pairs that the remote host initiates successful connection attempts to over the entire dataset capture period;

2. the number of distinct {local IP address, destination port} pairs that the remote host initiates unsuccessful connection attempts to over the entire dataset capture period; and

3. whether any local host initiates a connection attempt to the remote host.

Figure 1 shows the number of remote hosts for each number of distinct {local IP address, destination port} pairs that

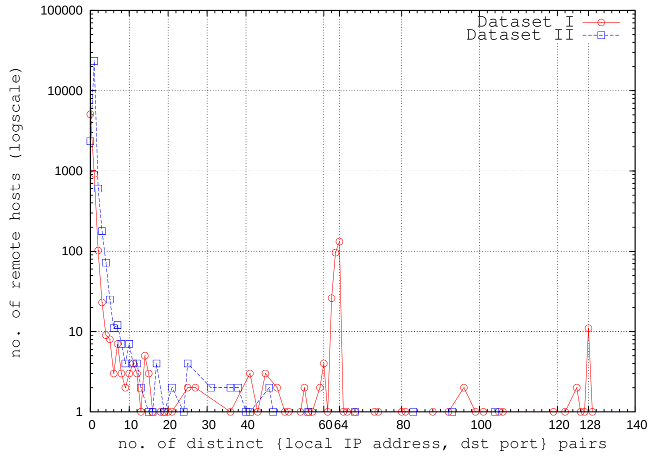

Figure 1: Number of remote hosts vs. the number of distinct {local IP address, destination port} pairs that these remotes initiated failed connection attempts to. (y axis in log scale; best viewed in color)

these remote hosts made unsuccessful connection attempts to (y axis in log scale). Approximately 78% (5,092 out of 6,562) and 9% (2,351 out of 26,859) of the remote hosts in the first and second datasets respectively, that initiated inbound connection attempts, made only successful connection attempts. Note that remote hosts which local hosts initiated outbound connection attempts to are excluded. The percentage of remote hosts that made only one unsuccessful connection attempt is approximately 14% (922 out of 6,562) and 88% (23,542 out of 26,859) in the first and second datasets respectively. In contrast, the percentage of remote hosts that made two unsuccessful connection attempts is only 1.6% and 2.3% in the first and second datasets respectively. While the decline in this percentage is sharp from one to two unsuccessful connection attempts, it is minor for more than two unsuccessful connection attempts. In fact, the percentage of all remote hosts that made two or more unsuccessful attempts is only 8.4% and 3.6% in the first and second datasets respectively. Note the two peaks at 64 and 128 in the x-axis, presumably due to scanners probing a range of IP addresses.

To see the distribution of remote hosts that made two or more unsuccessful connection attempts, Figure 2 plots the cumulative distribution for the number of remote hosts over the total number of distinct {local IP address, destination port} pairs that these remote hosts initiated failed connection attempts to (x axis in log scale). Note that more than 91% and 96% of remote hosts made at most one failed con-

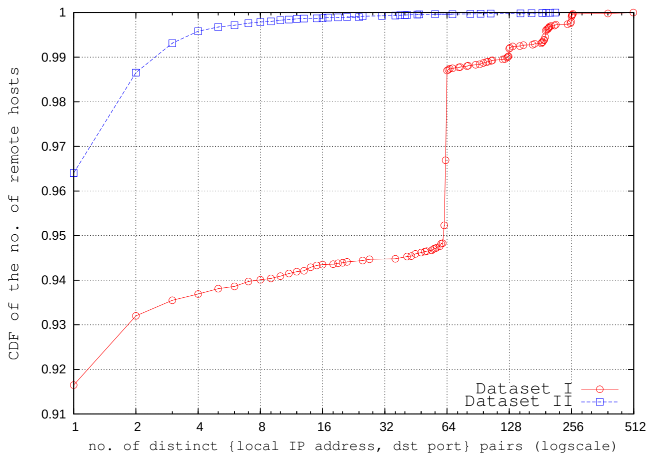

Figure 2: Cumulative distribution for the number of remote hosts and the total number of distinct {local IP address, destination port} pairs that these remotes initiated failed connection attempts to. (x axis in log scale; best viewed in color)

nection attempt in the first and second datasets respectively. However, while almost 93% did not make failed connection attempts with more than three local IP addresses (including those that made no failed connection attempts) in the first dataset, almost all remote hosts ($> 99\%$) did not in the second dataset. The variation between the two datasets in the distribution of the number of unsuccessfully contacted {local IP address, destination port} pairs for each remote host are due to several factors including: (i) the volume of scanning activity; (ii) the availability of offered network services; and (iii) the IP range of the monitored network.

Jung et al. [9] suggested using the ratio of the number of local hosts that a remote host unsuccessfully attempted to connect with vs. the total number of local hosts that the remote host contacted (either successfully or unsuccessfully) as a way to identify scanners. We define a similar ratio that also takes into account the contacted port; i.e., the number of distinct {local IP address, destination port} pairs that the remote host initiated failed connection attempts to vs. the total number of distinct pairs the remote contacted either successfully or unsuccessfully.

Figure 3 plots the cumulative distribution of this ratio for all remote hosts, excluding remote hosts that local hosts initiated outbound connection attempts to. The first observation is that the connection attempts for most remote hosts are either all successful or all unsuccessful. While this might seem a straightforward way to obtain a reference baseline in the absence of ground truth, we must differentiate between remote hosts that contact few or many local IP addresses. For example, in the second dataset, 96% of the remote hosts with a ratio of one (i.e., all their connection attempts failed) made only one failed connection attempt.

We employ these observations to derive a fine-grained classification of remote hosts based on the number of distinct {IP address, destination port} pairs that a remote host unsuccessfully contacted and the number of distinct {IP address, destination port} pairs that the remote host successfully contacted. For a remote host R, the following notation is used:

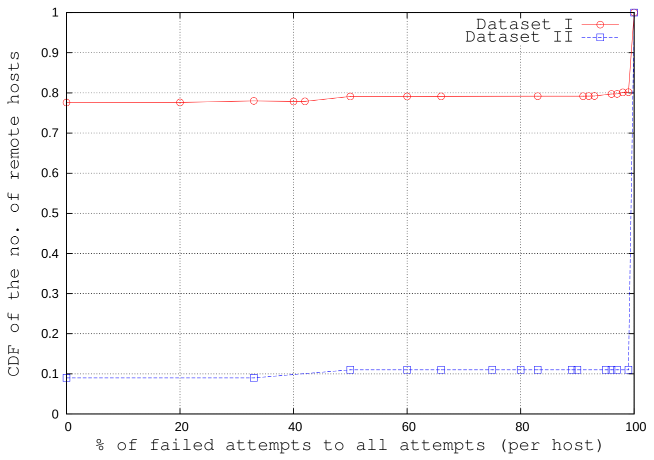

Figure 3: Cumulative distribution for the number of remote hosts and the ratio of the number of distinct {local IP address, destination port} pairs that each of these remotes initiated failed connection attempts to vs. the total number of distinct pairs the remote contacted either successfully or unsuccessfully.

$R_{inbound_S}$	the number of distinct {IP address, destination port} pairs the remote host initiated successful connection attempts to.
$R_{inbound_F}$	the number of distinct {IP address, destination port} pairs the remote host initiated failed connection attempts to.
$R_{outbound}$	the number of distinct {IP address, destination port} pairs that initiated connection attempts (whether successful or unsuccessful) to R.
R_ϕ	$R_{inbound_F}/(R_{inbound_S} + R_{inbound_F})$

Classification Criteria:

benign:	$((R_{outbound} \geq 1) \vee (R_{inbound_S} \geq 3)) \wedge (R_{inbound_F} \leq 1)$
likely benign:	$(R_\phi < 0.25)$
scanner:	$(R_{outbound} = 0) \wedge (R_{inbound_F} \geq 3) \wedge (R_{inbound_S} = 0)$
likely scanner:	$(R_{outbound} <= 1) \wedge (R_\phi \geq 0.75) \wedge (R_{inbound_F} \geq 2) \wedge (R_{inbound_S} \leq 2)$
unknown (one failed):	$(R_{outbound} = 0) \wedge (R_{inbound_F} = 1) \wedge (R_{inbound_S} = 0)$
unknown (others):	the remainder of remote hosts

The *benign* rule requires that R makes successful connections with at least three distinct {IP address, destination port} pairs and failed connection attempts with at most one pair. The host will also be considered benign if both at least one local host initiates an outbound connection attempt to R and R does not make failed connection attempts with more than one pair. R is classified as *likely benign* if it makes only successful connection attempts with at least 75% of distinct {IP address, destination port} pairs it contacts. This seems reasonable since remote hosts' traffic is monitored for a relatively long time (the dataset duration). Note that these rules are matched in order from *benign* to *unknown* such that if a remote host matches one category it will not be matched with the following rules.

The *scanner* rule applies to remote hosts that make only failed connection attempts with at least three distinct pairs. Also, there should be no outbound connections (whether successful or unsuccessful) made to these remote hosts from any local host. Such a strict heuristic is based on the assumption that it is unlikely for a benign remote host to make only failed connection attempts with three or more services in the monitored network, given that there is no outbound connections to the remote host. If R unsuccessfully contacts at least two distinct pairs and has a ratio of at least 0.75, then it is considered a *likely scanner*, even if it makes successful connections with up to two distinct pairs or if there is at most one outbound connection attempt to R (note that if $R_{inbound_S} = 2$, then $R_{inbound_F}$ must be 6). While the *scanner* heuristic captures the typical scanning pattern of probing non-existing network services (including stealthy scanners), the *likely scanner* heuristic captures fortuitous scanners that have found one or more active network services or that have been contacted by local hosts.

The first *unknown* rule is for remote hosts that make failed connection attempts with only one {IP address, destination port} pair and there is no outbound connections to them. Those remote hosts are hard to classify since failed attempts with one network service is not enough evidence of malicious intent. Causes of such cases include: (i) misconfiguration in the remote host; (ii) local servers or network failures; or (iii) very stealthy scanning (due, for example, the availability of many IP addresses for the scanner to scan from). The last *unknown* rule is for remote hosts that do not match any of the previous rules. It also includes backscatter traffic (e.g., the connection attempt starts with a SYN-ACK or a RST packet sent by the remote).

The probability of a scanner (with no prior knowledge of the targeted network) initiating a successful connection relies on the density of the offered services in the monitored network. Note that $R_{inbound_S}$ thresholds rely on the assumption that the density of the offered services (i.e., the number of open ports) in most of today's networks is usually very small with respect to the network's IP address range and the total number of possible services at each address (as it is the case in both datasets we study). The more local IP addresses offering the same network service (i.e., the same port number is open), the higher the probability of a scanner of this port making successful connections. To accurately evaluate successful inbound connection attempts in this case, each successful connection attempt is assigned a weight from 0 to 1 based on the density of the connection's destination port in the target network as follows:

$$weight = 1 - \frac{\text{number of local hosts with the port open}}{\text{number of local IP addresses}}$$

For example, if port 80 is open on 100 machines in a class C network, a successful connection attempt made to this port is given a weight of 1-(100/254) $\approx$ 0.6.

Classification	Dataset I		Dataset II	
Benign	(4.51%)	317	(4.52%)	1,308
Likely Benign	(72.34%)	5,086	(8.13%)	2,351
Scanner	(5.33%)	375	(1.05%)	304
Likely Scanner	(2.93%)	206	(1.6%)	464
Unknown (one failed)	(11.75%)	826	(79.9%)	23,109
Unknown (others)	(3.14%)	221	(4.79%)	1,386
Total		7,031		28,922

Table 2: Classification of remote hosts as a reference baseline.

Table 2 shows the classification results of both datasets. The variance between the two datasets in the percentages of remote hosts in each category is due to several reasons including: (i) the volume of scanning activities; (ii) the number of offered network services; and (iii) the volume of inbound traffic.

2.3 Challenges in Real-Time Scan Detection

In our analysis in the previous section, we had access to remote hosts' traffic over a relatively long period of time and there were no time or computational resources constraints. In the following, we discuss challenges involved in detecting scanners in real-time.

Detection Accuracy. The typical trade-off in intrusion detection between the rate of false alarm and the rate of detection is a challenging problem. The priority is to reduce false alarms while maintaining an acceptable detection rate. For a scan detector to have a reliable detection performance in terms of false and true positive rates over various environments, properties of the monitored network that may impact the detection must be considered by the scanning detection algorithm. It is also desirable to automate the process of setting the algorithm parameters so that the network administrator has minimal settings to manually configure.

Computational Resources. An accurate scan detection algorithm that consumes considerable resources of the monitoring system may not be applicable in practice. Therefore, it is important that the detector requires reasonable computational resources in terms of memory, processing time, and disk space. However, there is usually a trade-off between efficient use of monitoring system resources and reasonable accuracy of the detection algorithm.

Fast Detection. Post-detection responses, in general, are more effective if scanners are detected early. This requires making a decision upon observing a few number of failed connection attempts. However, the fewer the number of required observations of a host's behaviour, the less evidence of malicious intent is available. Hence, it is very challenging to set an appropriate trade-off between the false alarm rate and the number of connection attempts that scanners can perform before being classified as scanners.

Detecting Stealthy Scanners. To detect stealthy scanners [4], the state of external hosts must be kept for a long period of time (e.g., few days) after which the state is cleared, as long as there is no sufficient evidence to declare the remote host as a scanner. Thus, the memory footprint of the scan detection algorithm can easily increase to an unmanageable size. The challenge is to keep a state (of external hosts contacting the monitored network) that is as small as possible and to classify external hosts from as few connection attempts as possible.

Immunity to Evasion and Gaming. It is essential that the scan detection algorithm is as immune to evasion as possible, even for adversaries with a priori knowledge of the monitored network. It is also important to be resistant to DoS attacks where adversaries can manipulate the algorithm to flag innocent remote hosts as scanners.

3. LQS: ONLINE SCAN DETECTION ALGORITHM

Here we present the LQS scan detection algorithm. A description of the algorithm design and the algorithm pseudocode are given.

3.1 Overview

The algorithm depends on failed connection attempts as an indication of network scanning activity as discussed in Section 2. The LQS algorithm uses the exposure maps technique [25] as a decision oracle to determine whether a new connection attempt is potentially malicious. In this technique, a table of the services offered by a particular network is built automatically based on how internal hosts respond to incoming connection attempts. If a new connection attempt is destined to an entry in the services table, the connection is considered successful. Otherwise, the attempt is considered unsuccessful until its status is determined.

Unlike the exposure maps technique, however, LQS uses two tables (OPS and CPS; see *output* in Algorithm 1) that are updated continuously on-the-fly to keep the state of running services (i.e., open ports) in the monitored network: (i) the OPS table contains a list of active local network services; and (ii) the CPS table contains a list of local network services that were previously active and later became inactive (i.e., the most recent response from the corresponding port indicates that it is closed). Connection attempts to network services not in the OPS or CPS tables are immediately counted as scan events.

A remote host r is flagged as a scanner (i.e., inserted in the table S) in the following cases: (i) r has initiated unsuccessful connection attempts to at least k (default value 2) distinct local hosts (i.e., the case of horizontal or strobe scans); or (ii) at least $4k-3$ (i.e., $1+4(k-1)$) unsuccessful connection attempts are initiated by r to the same local host but on different destination ports (i.e., the case of vertical scans). In other words, r is flagged as a scanner if it has initiated failed connection attempts to at least k unique {IP address, port} pairs. The FC table contains counts of failed connection attempts for remote IP addresses that made at least one failed connection attempt.

If a local host initiates a connection to a remote host, the IP address pair is added to the whitelist CR such that failed connection attempts from the remote will not be considered if destined to the same local host (i.e., the remote's count in FC will not be increased as explained further below).

Each entry in the OPS, CPS, FC, CR, and S tables has a "write-expiry" interval such that the entry is deleted when the given period of time (I_1, I_2, I_3, or I_4) has lapsed since the last time the entry was inserted or modified.

The LQS algorithm does not flag remote hosts that make several successful connections as benign. Two advantages are gained by not whitelisting what appears as benign remote IP addresses: 1) avoiding possible evasion (see, e.g., [10]); and 2) quickly capturing a remote host change in state (i.e., being compromised).

3.2 Design Details

Pseudo-code of LQS is given in Algorithm 1. The function $NewConnection$ in line 2 returns true only if a new TCP or UDP connection is initiated (e.g., the first SYN packet from a remote host is seen for the TCP protocol). Note that if only SYN-ACK or RST packet is received from the remote, the connection will not be considered new to avoid backscatter traffic.

The $SuccessfulConnection$ function in line 22 returns true when the destination host responds positively to the source request indicating an open port (for the TCP protocol, a SYN-ACK packet indicates an open port). For each successful inbound connection (indicating an open port in the local network), the {local host IP address, destination port} pair is added to active network services table OPS or the corresponding entry is refreshed if it is already in OPS (line 23). If the pair exists in the CPS table, this means that the network service was previously available (i.e., was in the OPS) and then deleted from OPS and added to CPS, due to a previously rejected connection (RST packet) by the same pair. Therefore, in line 25, the corresponding entry is deleted from CPS. On the other hand, by receiving a RST packet from a previously open port (e.g., as a response to a TCP SYN packet) sent from a local host (as in line 39), indicating that the host is alive and the port is closed, the corresponding entry in the OPS table is moved to the CPS table. Entries in the CPS table are kept for a shorter interval I_2 ($I_2 << I_1$), as in lines 40 and 41.

For each remote host r contacting the monitored network, a counter ($FC[C.srcIP].count$) is updated for each new connection attempt as follows: if r attempts a connection (e.g., sending a SYN packet) to a local host l for the first time (where the contacted IP/port is not in the OPS table or the CPS table), this counter is incremented by one point (line 9). If r has previously contacted l and then attempts a new connection with l, but on a new destination port, then the counter is incremented by a quarter point (line 12). Note that each remote IP address (i.e., an entry in FC) is linked to a set of contacted local IP addresses ($FC[C.srcIP].Contacted$). Also, each local IP address in this set is linked to a set of destination ports contacted by the remote host ($FC[C.srcIP].Contacted[C.dstIP] .Ports$).

Making a connection attempt to a {local host IP address, destination port} pair contacted previously by the same remote host will not increase the remote host's counter. Only the first k unique {local host IP address, destination port} pairs are kept per remote host in the table FC. A remote host with a set of k entries is reported as a scanner and added to the table S, as in line 15. LQS returns true only if a new scanner is identified as in line 16.

Once a connection is successful, in addition to updating OPS and CPS accordingly, FC is updated as follows: 1) the contacted port is removed from the corresponding $Contacted$ list as in line 28; and 2) if r did not previously contact any other port in l ($Count(FC[C.srcIP].Contacted$ $[C.dstIP].Ports) = 0$) then r's counter is decremented by

```
INPUT:
    $C$ //a table of current connections
    $I_1$ (def=168 hr), $I_2$ (def=24 hr), $I_3$ (def=24 hr), $I_4$ (def=1 hr)
    $k$ (def=2) //number of unique (IP,port) pairs contacted unsuccessfully before declared as scanner
OUTPUT:
    OPS (global variable, def=$\emptyset$, expires after $I_1$)    // table of open ports (including their IPs)
    CPS (global variable, def=$\emptyset$, expires after $I_2$)    // table of previously open ports (RST seen)
    FC (global variable, def=$\emptyset$, expires after $I_3$)     // table of IP addresses with failed connections[1]
    CR (global variable, def=$\emptyset$, expires after $I_4$)     // table of {local host, contacted remote} pairs[2]
    S (def=$\emptyset$, expires after $I_3$)                       // table of scanners' IP addresses.
1  begin
2      if $NewConnection(C)$ then
3          if $IsLocalAddress(C.dstIP) \wedge [C.dstIP, C.dstPORT] \notin (OPS \cup CPS) \wedge [C.dstIP, C.srcIP] \notin CR)$ then
4              if $([C.srcIP] \notin FC)$ then add new entry for index $C.srcIP$ into $FC$
5              if $(FC[C.srcIP].count < k)$ then
6                  if $(C.dstIP \notin FC[C.srcIP].Contacted)$ then
7                      add new entry for index $C.dstIP$ into $FC[C.srcIP].Contacted$
8                      add new entry for index $C.dstPORT$ into $FC[C.srcIP].Contacted[C.dstIP].Ports$
9                      $FC[C.srcIP].count \Leftarrow FC[C.srcIP].count + 1$   //src,dst didn't contact previously
10                 else if $(C.dstPORT \notin FC[C.srcIP].Contacted[C.dstIP].Ports)$ then
11                     add new entry for index $C.dstPORT$ into $FC[C.srcIP].Contacted[C.dstIP].Ports$
12                     $FC[C.srcIP].count \Leftarrow FC[C.srcIP].count + 0.25$   //src,dst contacted previously
13                 end
14                 if $FC[C.srcIP].count = k$ then
15                     add new entry for index $C.dstIP$ into $S$
16                     return (True)
17                 end
18             end
19         else if $IsLocalAddress(C.srcIP) \wedge (C.dstIP \notin S) \wedge (RST \notin C.flags)$ then
20             add new entry for index $C.srcIP, C.dstIP$ into $CR$
21         end
22     else if $SuccessfulConnection(C) \wedge IsLocalAddress(C.dstIP)$ then
23         add $[C.dstIP, C.dstPORT]$ to $OPS$
24         if $[C.dstIP, C.dstPORT] \in CPS$ then
25             delete $[C.dstIP, C.dstPORT]$ from $CPS$
26         end
27         if $(C.dstPORT \in FC[C.srcIP].Contacted[C.dstIP].Ports)$ then
28             delete $FC[C.srcIP].Contacted[C.dstIP].Ports[C.dstPORT]$
29             if $(Count(FC[C.srcIP].Contacted[C.dstIP].Ports) > 0)$ then
30                 $FC[C.srcIP].count \Leftarrow FC[C.srcIP].count - 0.25$
31             else
32                 delete $FC[C.srcIP].Contacted[C.dstIP]$
33                 $FC[C.srcIP].count \Leftarrow FC[C.srcIP].count - 1$
34             end
35             if $([C.srcIP].count = 0)$ then
36                 delete $FC[C.srcIP]$
37             end
38         end
39     else if $(RejectedConnection(C)) \wedge ([C.dstIP, dstPORT] \in OPS) \wedge IsLocalAddress(C.dstIP)$ then
40         add add new entry for index $[C.dstIP, C.dstPORT]$ into $CPS$
41         delete $OPS[C.dstIP, C.dstPORT]$
42     end
43     return (False)
44 end
```

Algorithm 1: LQS (returns *True* when a new IP address is classified as a scanner)

one point (line 33); otherwise, r's counter is decremented by a quarter point (line 30).

If one or more packets with control flags set are missed due to network or host failures at either end, for a detector, an outbound connection may appear as either an inbound connection or as two connections: 1) an outbound connection; and then 2) an inbound connection during the lifetime of the same TCP or UDP flow. To overcome this limitation, LQS considers any remote host r that is not flagged as a scanner and that a local host has initiated a connection to as a non-malicious remote host (the corresponding IP address is kept in the table CR) for a time period determined by I_4, during which failed connection attempts initiated by r to the same local host will not be considered (as in lines 19 and 20).

[1] This table contains remote IP addresses having at least one failed connection attempt. Each remote IP address (i.e., an entry in the table) is linked to a set of unsuccessfully contacted local IP addresses. Each contacted local host is linked to a set of destination ports targeted by the remote host. Only the first k unique pairs are kept where remote hosts with sets of size k are considered scanners.

[2] Every local host in this table have sent a non-RST packet to the corresponding remote host.

3.3 Parameterization

Choosing an appropriate value for I_1 depends on the properties of the monitored network and the type of offered network services, where I_1 should reflect the approximate duration of inactivity, after which a network service is most likely being stopped or removed permanently from the monitored network (from various experiments on several sites, the one week default value appears appropriate). Similarly, the value for I_2 represents the expected duration of possible legitimate inbound activity after the port is closed (the default value of I_2 is one day).

The value of k should be set according to the stability and availability of the offered services in the target network. A higher value of k (than the default value) will result in fewer false positives since a remote host must make more first-contact failed connection attempts with local network services (i.e., contacting more {local host IP address, destination port} pairs) in order to be classified as a scanner. In contrast, the higher k is the greater the number of false negatives since scanners who contact fewer than k unique pairs within I_3 time window will not be reported. Given that a connection attempt destined to a pair in neither the OPS nor CPS tables is immediately considered a failure, even if the connection might be successful once a positive response is observed, setting $k = 1$ could yield a high false positive rate. In this case, the number of changes in the state of local hosts' ports from closed to open represents a lower bound on the number of false positives.

Testing on various traces from diverse network environments, we empirically determined a default value of 2 for k (this is also based on manual inspection of many samples). The reason that $k = 2$ represents a good threshold is because the probability that a benign remote host r contacts two local hosts on ports in neither the OPS nor CPS tables during I_3 time window is low. Therefore, given that failed connection attempts are inevitable, even in stable networks, a remote host making a failed connection attempt will be declared as a scanner only if it makes another failed connection attempt with a different local IP address. k can also be set to a number slightly above the median number of contacted local services by a single source address (e.g., the median + 1). In fact, k can be seen as a trade-off between fewer false positives and the ability to detect stealthy scans, or detect scanners faster from fewer connection attempts.

Scanners typically target a particular vulnerable port over a range of IP addresses, and thus unsuccessful connection attempts to the same local host are considered less malicious, even if destined to different ports. Therefore, by default, LQS flags a remote as a scanner only if it makes at least five failed connection attempts to the same local host but on different destination ports. This threshold is found empirically to provide fast detection of vertical scanners while significantly reducing the number of false positives.

3.4 Further Discussion

While IDS network sensors may skip packets that cannot be processed in real time, LQS keeps the state of open ports in the local network in the OPS and CPS tables so that a connection attempt that the scan detector missed one of its handshaking packets (e.g., uncaptured SYN-ACK packet) will not be interpreted as an unsuccessful connection. However, excessive skipping of packets by IDS sensors will increase the probability of generating false positives due to erroneously interpreting some outbound connections as inbound connections.

Setting up the scan detector behind the monitored network firewall leads to detecting only the scanning activity that made it through the firewall rules. Thus, the scan detector will capture more scanners if it is located at the gateway of the network. However, if the detector is located at the gateway, false alarms are expected for some network services. For example, in some applications (e.g., VoIP clients, IM, and P2P) a local host initiates a connection first to a server, which for some operations may request the client application in the local host to listen on a specific port for incoming connections initiated by other remote hosts for a specific period of time. Although the local host will open the required port, connection attempts from remote hosts to this port will fail if the network firewall is blocking inbound connections. Therefore, such failed connection attempts will appear as scanning activity.

To overcome this limitation, an active detector could send a TCP SYN packet (or empty UDP packet) directly to the target port without going through the firewall to find out whether the port is open or closed. If the port is open, failed connection attempts destined to this port must be ignored (i.e., not added to the FC table).

4. ADVANTAGES OVER TRW

This section illustrates the advantages of LQS over the TRW algorithm [9]. TRW classifies remote hosts as either benign, scanner, or pending according to the ratio of remote host's successful or unsuccessful connection attempts in the inbound network traffic within a specified time frame. The following metrics are compared for each algorithm.

Scan detection capability and the minimum number of connection attempts. While LQS can detect both horizontal (i.e., probing multiple IP addresses for the same port) and vertical scanning (i.e., probing a set of ports on the same IP address), TRW is designed to detect only horizontal scanning. For detecting a horizontal scanner, as a function of the TRW default parameters, TRW requires at least four consequent failed connection attempts initiated to four distinct local hosts within a given time window for a remote host to be classified as a scanner. In LQS, only two failed connection attempts initiated to two distinct local hosts are necessary to classify a remote host as a scanner. However, in case of vertical scanning, LQS requires five failed connection attempts initiated to five distinct ports in the same local host to classify a remote host as a scanner.

While the LQS algorithm will operate in a similar way to TRW (in the default setting) for detecting horizontal scanners when k is set to 4, first-contact successful connection attempts initiated by the scanners will not delay detection in LQS as in TRW. The fast detection in LQS makes it potentially suitable for fast post-detection responses.

False negative and false positive rates. In LQS, detecting scanners after their second failed connection attempt significantly decreases the false negative rate; i.e., the number of distinct IP addresses of scanners that were erroneously missed by the algorithm. Only those scanners that probed a single local host (and less than five distinct destination ports in this host) within I_3 time window will not be detected by LQS. In comparison, as a function of the TRW default parameters, TRW misses scanners that do not make four consecutive failed connection attempts within a given time window with no successful connection in between.

Given that hosts are usually configured using domain names and not IP addresses, causes of failed connection attempts from a benign remote host are often due to: (i) some of the contacted network services are temporarily unavailable; (ii) maintenance in the hosting servers; (iii) network failures; or (iv) outdated DNS entries. In LQS, failed connection attempts to previously offered services are not considered as the OPS and CPS tables keep track of previously open ports in the monitored network. Therefore, the high detection rate in LQS is not at the cost of high false positive rate (the same can also be inferred from our empirical evaluation on both datasets; see Section 5).

Suitability for worm detection. The TRW algorithm must wait for each new connection attempt to check whether it is successful or not. While a TCP connection status can be determined as successful after the remote host completes the 3-way establishment handshake, determining that the connection failed (in case of an unanswered connection due to a closed port, a non-existing host, or a firewall rule) might require waiting for a TCP timeout (two minutes is the default timeout value). For UDP, the fact that a local host responds with a UDP datagram to a remote host who initiated the exchange with a UDP packet indicates that the UDP port is open, and thus the connection is successful. Otherwise, if there is no UDP reply from the local host for a specific time (two minutes is the default timeout value) the connection is considered unsuccessful. Therefore, TRW is not designed to detect scanning worms that attempt to quickly propagate for which fast response is vital. Unlike the TRW algorithm, LQS does not wait for the connection state to be known; instead, it immediately assumes the connection is a failure if the {destination IP address, destination port} pair is in neither the OPS nor CPS tables.

Schechter et al. [21] proposed a hybrid approach that combines a variation of TRW and a credit-based connection rate limiting algorithm. The new variation detects fast scanning worms that can generate thousands of connection attempts (to find vulnerable machines) before being caught if only TRW is deployed for scan detection. Also, Jung et al. [8] proposed combining TRW with a rate-based sequential hypothesis testing algorithm that identifies if the rate at which a host initiates connections to new destinations is high. In addition to the drawback that limiting the rate at which first-contact connections can be initiated could block some legitimate hosts, these approaches are unable to detect stealthy worms.

Immunity to evasion and the ability to detect stealthy scanners. Since TRW must wait for a connection state to be known (two minutes is the default timeout value in TCP/UDP as discussed above), a single remote host can send thousands or millions of first packets (e.g., SYN packets) in the first two minutes to different local hosts and receives responses from open ports in the target network before being detected by TRW. If LQS is used, the remote host will be caught from the second connection attempt (e.g., immediately after sending the second SYN packet to a different local host).

Since TRW credits a remote host making successful connections by reducing its likelihood ratio towards being classified as benign, an adversary with knowledge of some available services in the target network can make successful connections to these services, while scanning the network to delay detection [10]. This feature in the TRW algorithm aims to avoid flagging a benign host that makes some failed connection attempts as a scanner. However, in addition to the possible evasion vulnerability, this feature is unnecessary in LQS to reduce the false positive rate since LQS takes into account various possible cases of benign failed connection attempts (i.e., those that have a high probability of not being a scan activity).

In the default setting, LQS is able to detect stealthy scanners after only two failed connection attempts to two distinct local hosts, even if the same remote host made successful connections before or between these failed attempts. In contrast, with the default parameters, TRW requires four consecutive failed connection attempts to classify a remote as a scanner. Also, the default time windows used in LQS to keep the state of the remote hosts are of longer duration than those used in TRW.

TRW has a list of friendly remote hosts similar to LQS non-malicious remote hosts table, CR. However, in TRW, if a remote host is added to the friendly list, any further connection attempts initiated by this remote to any local host will not be examined by TRW. Therefore, if a local host initiates a connection to a malicious remote host, the remote can scan the network without being detected. In LQS, only connection attempts to the same local host by the remote are not examined for possible scan activity. Therefore, a malicious remote that was previously contacted by a local host will only be able to scan the same local host without being detected.

Required Computing Time and Space In LQS, the number of entries in the OPS and CPS tables is bounded by the number of offered network services (during I_1 and I_2 time periods respectively) which is expected to occupy an insignificant amount of RAM (for example, < 5k in both the datasets studied; see Section 2.1). Both the LQS and TRW algorithms keep an individual set for scanners and also for non-malicious remotes that have been contacted by local hosts. TRW keeps an additional set for benign remotes.

The most expensive operations (e.g., insert and lookup) in LQS are those related to the FC table which contains remote IP addresses having at least one failed connection attempt. LQS keeps a list of up to k destination IP/port pairs a remote host unsuccessfully attempted to contact where the list is incremented only if the remote unsuccessfully contacts a new pair. Each remote IP address (i.e., an entry in the table) is linked to a set of unsuccessfully contacted local IP addresses. Also, in this set, each contacted local host is linked to a set of targeted destination ports by the remote.

Let L be the number of available local IP addresses and R be the number of remote hosts contacting the monitored network in a given time window. Also, let R_{failed} be a subset of R for those remotes making at least one failed connection attempt and $R_{success}$ be a subset of R for those remotes making at least one successful connection. Assuming that the used data structure needs 4 bytes to store one IP address and 2 bytes to store the port number, the *maximum* required space for FC in LQS is when every remote host in R_{failed} is vertically scanning a single local host: $R_{failed}((4+2)+(k-1)(4\times 2)) = R_{failed}(8k-2)$ bytes.

	Dataset I			Dataset II		
RB Classification	RB count	TRW	LQS	RB count	TRW	LQS
Benign	317	0	0	1,308	0	0
Likely Benign	5,086	0	0	2,351	0	0
Scanner	375	346	367	304	94	272
Likely Scanner	206	69	111	464	6	308
Unknown (one failed)	826	0	0	23,109	0	0
Unknown (others)	221	1	2	1,386	5	3
Total	7,031	416	480	28,922	105	583

Table 3: The distribution of the detected scanners by TRW and LQS among the categories of *RB*.

The table S requires at most $4R_{failed}$ bytes. Given that CR contains only active local hosts initiating outbound connections and that its write-expiry interval is short (one hour by default), the required space for CR is relatively small. For the default value $k = 2$, the maximum required space for LQS is approximately $18R_{failed}$ bytes. Therefore, the required RAM for LQS is bounded by a function which grows linearly with the number of remote addresses contacting the monitored network. The number of local IP addresses has no effect on the LQS RAM footprint (except the CR table).

In contrast, given that TRW requires that a remote host makes at least j (4, with the default parameters) consecutive failed attempts to j local hosts to be classified as a scanner (and likewise for benign hosts), the *minimum* required space for TRW is when the first j connection attempts for any given remote to unique local hosts are either all successful or all unsuccessful, and when the remote hosts in $R_{success}$ contact only one local host. TRW stores {remote IP address, local IP address} pairs for both successful and failed inbound connection attempts ($8R_{success}$ bytes, and $(8j)R_{failed}$ bytes), a table of scanners' IP addresses ($4R_{failed}$ bytes at most), a table of benign remotes (for small space, but complex to compute precisely, the required space is omitted), a table of remotes' IP addresses that have been contacted by local hosts (similar to CR in LQS, we omit the space required for this table), and a table of likelihood ratios of remotes that contact the monitored network (requiring $(4+2)R$ bytes; assuming 2 bytes to store the ratio). For $j = 4$, the *minimum* required space for TRW is then: $36R_{failed} + 8R_{success} + 6R$ bytes.

Therefore, the *maximum* required memory footprint for LQS is smaller than the *minimum* required for TRW. Also, in practice, a significant percentage of remotes (including benign and scanners) are expected to make both successful and failed connection attempts, and thus L will have an effect on the required space by TRW. Notice that while TRW must keep a state for each remote that initiates a connection attempt (whether successful or failed) to the local network, LQS keeps a state only for remotes that initiate failed connection attempts.

Both algorithms must be called for each new connection attempt. In LQS, the most expensive operation is the lookup operation in the FC table. The processing time for such lookup (and insertion operation if required) depends on the data structure used and the number of entries. The ideal data structure to lookup entries in LQS tables is a hash table. The most expensive lookup in the TRW algorithm is to determine if the destination IP address has previously contacted the source IP address. In both algorithms, if hash tables are used, the computational cost is constant for one call of the algorithm and the number of calls is linear to the number of inbound connections.

5. EMPIRICAL EVALUATION

We have implemented LQS in the Bro language (Bro 1.4 NIDS [1]) and used the TRW implementation of Bro. For the purpose of comparison with LQS, the TRW algorithm was configured to monitor remote hosts' behaviour over a one day time window (similar to LQS) rather than the 30 minutes default value. While this configuration enables the TRW algorithm to detect more stealthy scanners, it increases the required memory footprint. The write-expiry interval in TRW detected scanners list was removed to keep track of all detected scanners by TRW over the entire dataset capture period. Using our reference baseline (see Section 2.2), we measure the performance of both algorithms using the following metrics:

1. True Positive Rate (i.e., detection rate): is the proportion of the distinct IP addresses of scanners that are correctly reported by the detector:

$$\textbf{TP rate} = \frac{\text{no. of true pos.}}{\text{no. of true pos.} + \text{no. of false neg.}}$$

2. False Positive Rate: is the proportion of the distinct IP addresses of non-scanners that are erroneously reported as scanners by the detector:

$$\textbf{FP rate} = \frac{\text{no. of false pos.}}{\text{no. of false pos.} + \text{no. of true neg.}}$$

3. Efficiency: is the proportion of the reported scanners by the detector that are true positive:

$$\textbf{Efficiency} = \frac{\text{no. of true pos.}}{\text{no. of true pos.} + \text{no. of false pos.}}$$

For any intrusion detector, if the number of true negative samples is significantly larger than the number of true positive samples, the FP rate is expected to be small, regardless of the detector performance, and therefore calculating the *efficiency* (or the *false discovery rate* which is 1−efficiency) is a more meaningful performance metric than the FP rate. Similarly, if the number of true positive samples is significantly larger than the number of true negative samples, the TP rate is expected to be large, regardless of the detector performance, and thus calculating the *false omission rate* (i.e., no. of false negatives/(no. of false negatives + no. of true negatives)) is a more meaningful performance metric than TP rate. Based on the datasets studied in the literature (e.g., [3]) and our datasets, network scanning activity is often of the former case.

Table 3 shows the distribution of TRW and LQS detected scanners among the categories of our reference baseline. TRW detected 416 and 105 scanners, whereas LQS detected 480 and 583 scanners in the first and second datasets respectively. None of the remotes in the benign or likely benign

Performance Metrics		Dataset I		Dataset II	
		TRW	LQS	TRW	LQS
RB_1	TP rate	0.7143	0.8227	0.1302	0.7552
	FP rate	0.0002	0.0003	0.0002	0.0001
	Efficiency	0.9976	0.9958	0.9524	0.9949
RB_2	TP rate	0.9227	0.9787	0.3092	0.8947
	FP rate	0.0002	0.0003	0.0002	0.0001
	Efficiency	0.9971	0.9946	0.9495	0.9891

Table 4: Performance evaluation.

categories were marked by any of the algorithms as a scanner. Also, remotes that made only one failed connection attempt were not flagged as a scanner since both algorithms require more than one connection attempt for any given remote host. False positives in both algorithms appeared only in the unknown (others) category of the reference baseline.

Table 4 shows the performance of both algorithms according to the metrics discussed above as follows:

i) $\boldsymbol{RB_1}$**:** remote hosts in both the scanner and the likely scanner categories are true positives and the remainder are true negatives (i.e., the benign, likely benign, and unknown categories); and

ii) $\boldsymbol{RB_2}$**:** true positives are only those in the scanner category and true negatives are those in the benign, likely benign, and unknown categories (this is a more relaxed reference baseline where detected scanners from the likely scanner category are ignored; note that the likely scanner category is not added, neither to the true positives nor to the true negatives).

Dataset I and $\boldsymbol{RB_1}$. LQS demonstrated a better TP rate than TRW by more than 15%. As expected, the FP rate in both algorithms is very low because of the significantly large number of samples relative to the number of true positives. The detection efficiency is high in both algorithms (less than 1% of detected scanners are false positives).

Dataset 1 and $\boldsymbol{RB_2}$. TP rate is improved in both algorithms where LQS is better by only 6%. The efficiency is also high in both algorithms. Therefore, both algorithms achieved good performance in detecting the entries in the scanner category which represents remotes that significantly exhibit scanning rather than normal behaviour (as discussed in Section 2.2).

Dataset 2 and $\boldsymbol{RB_1}$. TRW detected 13% of scanners in the second dataset. In contrast, LQS has a detection rate of 76% while maintaining a slightly smaller (better) FP rate, and efficiency better than TRW by approximately 5%.

Dataset 2 and $\boldsymbol{RB_2}$. Even with RB_2, the TRW detection rate (TP rate) is only 31%. LQS performed better both in detection rate (90%) and efficiency (0.99 vs. 0.95).

In the second dataset, many scanners initiated few connection attempts and had a low scanning rate, which contributed to the poor performance of TRW. This reflects the current trend of stealthy probing by scanners (e.g., as noted by Allman et al. [3]), perhaps due to the large number of IP addresses (e.g., infected hosts) involved in some coordinated scanning campaigns. For example, rather than the conventional aggressive scanning behaviour of many typical worms, stealthy scanning activity is now more common (e.g., by stealthy worms and bots [14, 15]).

6. RELATED WORK

Network Security Monitor (NSM) IDS [7] examines the destination IP addresses contacted by a remote host, where the remote is considered anomalous once it contacts more than 15 local hosts within an unspecified time window, or when the remote attempts a connection to a non-existing host. The scan detection scheme in GrIDS [23] is another early approach that graphically shows remotes' activities and connectivity over time where a graph of one remote contacting many local hosts could indicate a possible scan activity. Kato et al. [11] proposed a real-time IDS for detecting network attacks. They set a threshold of the number of TCP ACK/RST packets returned to the same remote within a specified time window after which the remote host is labeled as a scanner.

A probabilistic model used by Leckie and Kotagiri [13] considers both the number of local hosts or ports accessed by a remote, and how unusual these accesses are. The model gives a connectivity probability for each local host and each port to rate the likelihood of a given remote being benign or scanner. This approach requires sufficient knowledge of the monitored network and dynamic updates of the hosts/ports probabilities according to the network changes. Spice [22] is a port scan detector designed for stealthy scans using a statistical model such that packets sent to rarely accessed IP address/port combinations are considered more anomalous, and thus have a higher probability to exceed an adjustable anomaly score threshold where an alert is generated.

The system introduced by Robertson et al. [18] gives each remote host a score based on the number of unique destination IP/port pairs of failed connection attempts such that a remote host is classified as a scanner if its score is greater than an empirically derived alert threshold. Using a statistical model, Kim et al. [12] calculate a normal distribution of destination IP addresses/port pair in a network and then use various statistical tests to analyze traffic rates to detect port scans.

A scan detection preprocessor plug-in called sfPortscan [19] in Snort [20] generates an alert when a remote host attempts to connect to more than a predefined threshold of local hosts (four IP addresses is the default threshold) or to more than a predefined threshold of ports (19 is the default) within a predefined time window (one minute is the default). This method is similar to the tools developed by Fullmer et al. [5] and Navarro et al. [17]. However, by not exceeding the probing threshold within the specified time window, an adversary can easily evade detection.

TRW [9] is implemented as a Bro policy such that scanners' traffic can be dropped by setting the appropriate interface between Bro and the corresponding network router. A simplified variant of TRW that required less memory footprint and can detect vertical scanning is proposed by Weaver et al. [24]. The authors also proposed a suppression algorithm for worm containment with dynamically adjustable thresholds (see Section 4 for additional related work on using TRW for worm detection). A similar modification [16] considered vertical scanning and extended TRW to detect UDP and ICMP scans. A Bloom filter is used to filter the input to TRW so that only unique source and destination IP addresses, destination port, and protocol are processed by TRW. The TRW time window of keeping the state of remote hosts' connection attempts is also increased to detect stealthy scanners.

7. CONCLUDING REMARKS

Network scanning remains a useful reconnaissance activity by attackers. Given the high ability of compromised machines in today's Internet, scanning which is highly distributed specifically in order to achieve stealthiness [6] is now recognized as a feasible and practical strategy to avoid triggering IDSs. Also, post-detection responses to network scanning often require fast and accurate detection.

LQS specifically addresses these issues, as a real-time network scanning detector that detects stealthy scanners quickly, while achieving high detection rates and very low false positive rates in comparison to the TRW algorithm. Moreover, LQS requires a smaller memory footprint and has a higher immunity to evasion. We also presented a novel methodology to obtain an estimated ground truth for evaluating network scanning detectors.

8. ACKNOWLEDGMENT

We thank J. Jung, anonymous reviewers, and members of the Carleton Computer Security Lab (CCSL) for helpful comments. The second author is Canada Research Chair in Authentication and Computer Security and acknowledges NSERC for funding the chair, and a Discovery Grant. Partial funding from NSERC ISSNet is also acknowledged.

9. REFERENCES

[1] Bro intrusion detection system. Accessed: May 2010. `http://bro-ids.org/`.

[2] Ethereal display filter reference. Accessed: Aug 2010. `http://www.ethereal.com/docs/dfref/`.

[3] M. Allman, V. Paxson, and J. Terrell. A brief history of scanning. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 77–82, 2007.

[4] CERT. Advanced scanning. CERT Incident Note IN-98.04 (Sept. 29 1998). `http://www.cert.org/incident_notes/IN-98.04.html`.

[5] M. Fullmer and S. Romig. The OSU Flow-tools package and Cisco Netflow logs. In *Proceedings of the 14th Systems Administration Conference (LISA'00)*, pages 291–303, New Orleans, LA, USA, 2000. USENIX Association.

[6] C. Gates. Coordinated scan detection. In *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS'09)*, February 2009.

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

[8] J. Jung, R. A. Milito, and V. Paxson. On the adaptive real-time detection of fast-propagating network worms. In *Proceedings of the 4th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'07)*, pages 175–192, Lucerne, Switzerland, 2007. Springer-Verlag.

[9] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy*, May 2004.

[10] M. G. Kang, J. Caballero, and D. Song. Distributed evasive scan techniques and countermeasures. In *Proceedings of the Fourth GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'07)*, Lucerne, Switzerland, July 2007.

[11] N. Kato, H. Nitou, K. Ohta, G. Mansfield, and Y. Nemoto. A real-time intrusion detection system (IDS) for large scale networks and its evaluations. *IEICE Transactions on Communications*, E82-B(11):1817–1825, 1999.

[12] H. Kim, S. Kim, M. A. Kouritzin, and W. Sun. Detecting network portscans through anomaly detection. In *Proceedings of Signal Processing, Sensor Fusion, and Target Recognition XIII*, pages 254 – 263, 2004.

[13] C. Leckie and R. Kotagiri. A probabilistic approach to detecting network scans. In *Proceedings of the Eighth IEEE Network Operations and Management Symposium (NOMS'02)*, pages 359–372, 2002.

[14] Z. Li, A. Goyal, and Y. Chen. Honeynet-based botnet scan traffic analysis. In *Botnet Detection*, pages 25–44. Springer US, 2008.

[15] Z. Li, A. Goyal, Y. Chen, and V. Paxson. Automating analysis of large-scale botnet probing events. In *ASIACCS*, pages 11–22, 2009.

[16] V. Nagaonkar. Detecting stealthy scans and scanning patterns using threshold random walk. *Master's thesis, Dalhousie University, Canada*, 2008.

[17] J.-P. Navarro, B. Nickless, and L. Winkler. Combining Cisco netflow exports with relational database technology for usage statistics, intrusion detection, and network forensics. In *the 14th Systems Administration Conference (LISA'00)*, pages 285–290. USENIX Association, 2000.

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

[19] D. Roelker, M. Norton, and J. Hewlett. sfPortscan. Sept. 2004.

[20] M. Roesch. Snort: lightweight intrusion detection for networks. In *Proceedings of the 13th Systems Administration Conference (LISA'99)*, pages 229 – 238, Seattle, WA, USA, 1999. Usenix Association.

[21] S. E. Schechter, J. Jung, and A. W. Berger. Fast detection of scanning worm infections. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection*, pages 59–81, 2004.

[22] S. Staniford, J. A. Hoagland, and J. M. McAlerney. Practical automated detection of stealthy portscans. *Journal of Computer Security*, 10(1/2):105–136, 2002.

[23] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagl, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS - a graph based intrusion detection system for large networks. In *Proceedings of the 19th NISSC*, pages 361–370, 1996.

[24] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms, revisited. *Malware Detection (Advances in Information Security)*, 27:113–145, 2007.

[25] D. Whyte, P. C. van Oorschot, and E. Kranakis. Tracking darkports for network defense. In *Proceedings of ACSAC*, pages 161–171, 2007.