DEFENSES AGAINST NETWORK SCANNING AND OTHER MALICIOUS REMOTE HOST ACTIVITY: EMPIRICAL STUDIES, ANALYSIS, AND NEW APPROACHES

by

Mansour Alsaleh

A thesis submitted to
the Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario
November 2011

Abstract

Network resources linked to the Internet are susceptible to a variety of attacks that become increasingly hard to detect with the increasing complexity in Internet traffic dynamics and heterogeneity. Even with the assumption that end-to-end Internet dynamics can be correctly characterized, it is exceptionally difficult to identify malicious network traffic, as it may be crafted to adhere to network protocol specifications both syntactically and semantically and to mimic legitimate traffic behaviour. For example, reconnaissance activities may be so crafted, with the objective of gathering information to launch subsequent attacks.

In this thesis, we focus on network scanning and automated password guessing attacks, two types of widespread malicious network activity that are known precursors to a broad range of compromises of machines and accounts. Recently, these activities are often conducted in a large-scale capacity targeting apparently random networks, rather than being directed or strategic. We conduct an analytical and empirical study of these two malicious activities using recent real-world network traces and logs collected at various sites. We examine and evaluate selected detection and prevention approaches to identify their limitations and strengths. For network scan detection, our results show that there is often a crucial trade-off between detection and false positive rates, due to a lack of both a built-in algorithmic adaptability and a manual parameterization criterion based on the deployment environment. For password guessing attacks against existing login protocols, we find there is a fundamental trade-off between user login convenience and login security with respect to password guessing.

To address the limitations found, we introduce two novel network scan detection algorithms and a new password guessing resistant protocol. Our empirical evaluation argues that they offer practical defenses against such malicious network activity. Our detection and defense mechanisms are designed to capture large-scale events and those launched by adversaries with access to a large number of machines (e.g., a botnet).

As part of our empirical evaluation, to our knowledge, we are the first to explore in detail the problems that can arise when evaluation is based on a ground truth reference rather than absolute ground truth. We model the problem of evaluating detection algorithms in the absence of absolute ground truth, and analyze the requirements of using a ground truth reference for either evaluating one intrusion detector or comparing multiple detectors.

Acknowledgements

I would first like to thank my advisor, Prof. Paul Van Oorschot, for his support, guidance, patience, and always prompt, detailed comments throughout this process.

I also want to thank the many people whose comments have helped to improve this work including: the co-authors of the publications I worked on during my PhD; the members of my committee, Professors Michel Barbeau, Guy-Vincent Jourdan, Scott Knight, and Ashraf Matrawy for helping improve this thesis with their expertise and guidance; my colleagues in the Carleton Computer Security Lab (CCSL) that I had related discussions with; the anonymous journal/conference referees who provided constructive feedback; my friends, especially Deborah, who reviewed or proof-read any part of this thesis; and Prof. John McHugh for his detailed discussions and useful comments, from which I learned much, though any errors remaining in this thesis are my own.

I am very thankful to my family, especially my parents, for their support and encouragement.

Finally, I want to acknowledge funding received from several sources including: NSERC ISSNet, Carleton University, and the Saudi Cultural Bureau in Canada.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

AGT Absolute Ground Truth

ATT Automated Turing Test

CAPTCHAs Completely Automated Public Turing test to tell Computers and Humans Apart

DHCP Dynamic Host Configuration Protocol

DOS Denial-of-Service

EM Exposure Maps

GTR Ground Truth Reference

IDS Intrusion Detection System

ISP Internet Service Provider

LQS Lightweight, Quick and Stateful Algorithm

NAT Network Address Translation

P2P Peer to Peer

PGRP Password Guessing Resistant Protocol

STRW Stateful Threshold Random Walk

TRW Threshold Random Walk

Chapter 1

Introduction

In this thesis, we both analyze and conduct several empirical studies on two widespread types of malicious network activity that are known precursors to a broad range of compromises: (i) network scanning; and (ii) automated password guessing attacks. The majority of the thesis is on the first. We evaluate existing defense mechanisms, introduce two novel network scan detection algorithms and a new password guessing resistant protocol. In the network scanning side, we also model the problem of performing evaluation in the absence of absolute ground truth and we present an evaluation approach to address uncertainties in a ground truth reference.

1.1 Motivation and Scope

With network complexity continuously increasing and Internet traffic heterogeneity, scale, and rapid change in dynamics, distinguishing malicious network traffic from legitimate traffic is an escalating challenge. New vulnerabilities in network protocols and services are disclosed on a daily basis. Even well-administered networks are vulnerable to zero-day exploits. Also, some secure network services become vulnerable only when offered simultaneously with each other. For an outside attacker, the common objective is to gain access to a target host and then exercise the privileges accessible on the target's system, either to leverage its system resources for performing illegal activities (e.g., sending spam) or to acquire information that should have been protected (e.g., collect users' sensitive information).

Querying and mining accessible network resources through reconnaissance and automated techniques can be a great advantage to adversaries either when searching for machines over the Internet containing software vulnerabilities that can be exploited, or to acquire users' login credentials to bypass access control mechanisms. Stealth

and mimicking legitimate network activity are two important features to avoid detection during such malicious network activity that represents a reconnaissance phase. Recently, these activities are often conducted in a large-scale capacity targeting apparently random networks, rather than being directed or strategic. In this thesis, we study in depth two types of widespread malicious network activity: (i) network scanning; and (ii) automated password guessing attacks.

Network scanning is the process of attempting 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. While network scanning has legitimate uses such as vulnerability scanning of one's own internal machines, it is often an indicator of malicious activity. In fact, scanning is a valuable reconnaissance step that precedes many of today's Internet attacks. Network scanning reveals valuable information about accessible hosts over the Internet and their offered network services, which allows significant narrowing of potential targets to attack, by locating hosts running vulnerable network services. 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 to servers.

In targeted attacks, adversaries commonly use automated scanning tools (e.g., NMAP [79]) to gather information about the targeted network (e.g., offered services and associated software versions) [81]. In addition, many network worms scan various Internet subnets to locate further vulnerable machines to attack [95, 118]. Likewise, scanning is an effective way for a botnet to recruit new bots [59, 60]. Scanning also remains a common technique used to determine if a remote site has a particular security vulnerability in order to use it to host phishing websites [67]. Responses to scanning can also reveal useful information about the structure of the target network and firewall policies. In fact, scanning traffic represents a significant portion of inbound traffic for most networks accessible through the Internet [6].

Network scanning activity raised attention of network administrators back in 2001 with the Code Red and Nimda worm outbreaks where scanning activity outweighed legitimate traffic in some networks in terms of the number of connection attempts [6].

In a study of Internet scanning activity from 1994 to 2006, Allman et al. [6] noticed a peak of scanning activity in the years 2003 and 2004 due to aggressive scanning worms (e.g., MyDoom, Sasser, Welchia, and Bobax). The number of scanners and detected scanning activity declined gradually after 2004 until the end of the collected data (end of 2006) for that study as a result of more efficient and careful scanners operating with a lower profile to avoid detection. Although numerous network scanning detection approaches have been proposed in literature (e.g., [105, 89, 50, 45, 97, 102, 36, 101]), very few proposals offer both reasonably accurate and efficient detection. The high false positive and false negative rates inherent with these techniques have contributed to few being adopted by intrusion detection systems (IDSs) and they are rarely used to automatically block identified scanners.

For several reasons, some network administrators may argue that network scanning detection is unnecessary. First, false alarm rates could be high due to having some benign network activities that resemble scanning behaviour in terms of the features used for detection. Second, given that it is easy to spoof a source IP address, an adversary can make the detection system flag benign IP addresses as scanners. Third, a significant amount of scanning activity could be linked to relatively old exploits where systems running targeted services are most likely patched. Therefore, such administrators believe that detecting scanners is a hard problem and it is better to focus on tightening the security of their networks. For example, a good defensive measure is a well configured and maintained firewall with strict rules to help block all scanning activities, except scanning traffic destined to publicly accessible network services.

On the other hand, some network administrators believe that network scanning detection could help them in securing their networks in various ways: (i) scanning activity is a known precursor to network attacks that might be mitigated by blocking detected scanner IP addresses (e.g., scanning worms and bots); (ii) inbound and outbound traffic from and to remote hosts identified as scanners can be monitored closely by a network IDS; (iii) logs of scanning activity may help administrators to focus on patching the most targeted network services (or alternatively to hide services on obscure ports or shut them down temporarily in some cases); (iv) detecting

local scanners helps network administrators in finding compromised hosts within their networks; and (v) blocking scanning traffic will reduce the load on internal hosts and network devices. Further discussion on the relevance and importance of detecting external scanners is given in Sections 2.1, 2.4, 2.5, 4.1, and 5.1.

The majority of proposed scanning detection techniques depend on detecting abnormal network traffic in remote host traffic directed to the local network. Unfortunately, most proposed detection features can be evaded easily by informed adversaries to avoid detection [43]. In fact, the only exception appears to be features based on a remote host's successful or failed connection attempts (e.g., [45, 124]). The reason seems to be that the objective of network scanning is to find open ports, and thus it is assumed that the adversary does not know the open ports in the monitored network. Nevertheless, under the assumption that the detection mechanism is known, scanners may elude detection by scanning slow enough to avoid triggering threshold-based detection alarms. Moreover, some advanced scanners with access to a large number of scanning hosts can divide the target IP range among these hosts so that each host can scan some small number of IP addresses before being detected and possibly blocked.

Most post-detection responses (e.g., limiting the amount of information the scanner can learn about the monitored network by blocking some of their inbound network traffic) require fast, real-time detection of scanners. However, there is a trade-off between detection accuracy and fast detection. It is also challenging to find the right trade-off between detection and false alarm rate, as a lower false positive rate usually leads to a lower detection rate.

Some other techniques aim to limit the effectiveness of network scanning by changing the TCP or IP protocols so that either an authentication is required first to contact a network service or that by increasing the number of ports or IP addresses, random IP address/port space probing could be made ineffective. For example, port knocking [54] requires changing the TCP/UDP protocols so that a remote host must first send a TCP/UDP packet to a sequence of ports (as a password) before contacting the network service in question. Also, it has been suggested [23] that by using only the much larger default 64-bit subnet address space of IPv6, network scanning will be costly for scanners (see Section 8.5 for further discussion). In this thesis, however,

we focus on practical solutions that do not require changing the current Internet infrastructure. Also, we only focus on the currently dominant IPv4 networks.

A second type of malicious network activity that we examine in this thesis is automated password guessing attacks. Guessing attacks are highly effective if a reasonable number of password guesses are permitted by the login system. SANS [94] identified password guessing attacks on websites as a top cyber security risk. Defending against such guessing attacks becomes more challenging over time as many of these guessing attacks are now launched by adversaries with access to a large number of machines (e.g., a botnet). Most existing defense approaches either negatively affect usability (i.e., user login convenience) or suffer a crucial trade-off between usability and login security with respect to password guessing.

1.2 Thesis Statement and Hypotheses

The major goals of this research can be summarized as follows:

G1. gaining a better understanding of the nature of, and motives behind, the malicious network activity discussed above using real-world network traces;

G2. analyzing, empirically evaluating, and comparing existing defense mechanisms;

G3. improving selected existing defensive approaches and providing new practical defenses based on our analysis and understanding of these malicious activities.

Given the above discussion of the problem and the limitations of the existing defensive approaches, the research in this thesis pursues the following hypotheses.

Hypothesis 1. We hypothesize that incorporating selected properties of mainstream scan detector's operational environment into operational parameters of the scan detector will improve the detection accuracy in terms of true and false positive rates; and that it is possible for some scan detection algorithms to automate the process of setting these parameters rather than relying on the network administrator to manually choose appropriate values.

Hypothesis 2. It is possible to design a scan detector that simultaneously provides high detection accuracy, fast detection speed (the speed is measured in terms of the

number of connection attempts that a scanner can perform before being detected), and efficient use of monitoring system resources. To be more specific, by this we mean, to design a scan detection algorithm which performs better in respect to this criterion than existing algorithms which arguably have limitations with respect to one or more of these properties.

Hypothesis 3. For a scan detector based on the absolute number of a remote host's successful or failed connection attempts, the false positive rate can be significantly reduced by designing the detector to take into account the various possible causes of benign failed connection attempts which should not be considered as scanning activity.

Hypothesis 4. Some scan detectors are more appropriate for certain types of network environments than others, and we can identify these environments for several scan detectors studied.

Hypothesis 5. In real-world network traffic, typically many events can be equally interpreted as legitimate or intrusions and therefore establishing absolute ground truth (AGT) is infeasible since it depends on unknowable intent. An estimated ground truth based on a discrete classification criteria (i.e., grouping events into positive and negative classes) can be misleading since typical detection accuracy measures are strongly dependent on the chosen criteria. This motivates our next hypothesis. There are approaches that can be taken to address and model uncertainties in a ground truth reference (GTR) in the absence of AGT, either in establishing a GTR for a given network trace or in comparing one or more detectors with the GTR.

Hypothesis 6. It is possible to design a more effective and more general password-based login system with a threat model of large-scale password guessing attacks in mind. Related to this hypothesis, we are interested to determine if it is possible to significantly restrict such attacks without being at the expense of user login convenience, or if the often believed trade-off between user convenience and login security with respect to password guessing is inevitable.

1.3 Main Contributions

As discussed in the opening paragraph of this chapter, in this thesis we both analyze and conduct several empirical studies on two widespread types of malicious network activity that are known precursors to a broad range of compromises of machines and accounts: (i) network scanning; and (ii) password guessing attacks (e.g., brute force and dictionary attacks). For network scanning, we focus on scans initiated by remote scanners targeting the monitored network. We introduce two novel network scan detection algorithms and a new password guessing resistant protocol. Our empirical evaluation shows that they represent practical defenses against such malicious network activity. We also model the problem of performing evaluation in the absence of AGT and we present an evaluation approach to address uncertainties in a GTR. The main contributions of this thesis can be summarized as follows.

1. **Empirical Studies to Gain a Better Understanding of Malicious Network Activity**

   (a) **Network Scanning.** We conduct an in-depth study of two known network scanning detection techniques, the TRW [45] and EM [123] algorithms. The study consists of an analytical and empirical analysis and comparison in terms of detection accuracy, detection speed, computational resources, scalability, and resistance to evasion and attacks. The empirical evaluation is conducted on several recent datasets collected at various sites of different nature and size. We identify the limitations and strengths of these techniques and we show that there is a crucial trade-off between detection and false positive rates, due to lack of both a built-in algorithmic adaptability and a manual parameterization criterion based on the properties of the monitored network environment. We also provide guidelines on parameterization and configuration of both algorithms to enhance detection accuracy.

   (b) **Password Guessing Attacks.** We conduct the first reported empirical analysis of ATT-based (automated turing tests; e.g., CAPTCHAs) login protocols (e.g., [84, 113]) that are designed to address large-scale online

password guessing attacks (e.g., brute force and dictionary attacks from a botnet of up to tens or hundreds of thousands of nodes). We used two datasets from an operational network environment, where each dataset logs events of a particular remote login service, over a one-year period each. Our study shows that there is a fundamental trade-off between user login convenience and login security with respect to password guessing in existing login protocols; i.e., increasing the number of ATTs to limit password guessing attempts also increases the number of ATTs legitimate users must answer.

2. **LQS.** We propose LQS, a new lightweight network scan detection algorithm that detects scanners as early as from their second connection attempt to the monitored network. Unlike previous scan detection approaches (e.g., [45, 90]), LQS keeps the state of offered network services over time to evaluate inbound connection attempts. 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 [45]. For the datasets and environments examined, our empirical analysis shows significant improvements over TRW in the following key properties: (i) fast detection of scanning activity to enable prompt response by IDSs; (ii) acceptable rate of false alarms, keeping in mind that false alarms may lead to legitimate traffic being penalized; (iii) high detection rate with the ability to detect stealthy scanners; (iv) efficient use of monitoring system resources; and (v) immunity to evasion.

3. **STRW.** We confirm that TRW [45] was designed for scan detection in a controlled enterprise network environment, identifying several causes of false positives in now increasingly popular transient network environments, i.e., where legitimate network services appear only intermittently at a given IP address with existing and new network hosts free to join and leave the network at anytime. Accordingly, we present a modified algorithm (STRW) for scan detection which takes into account the identified causes of TRW false positives in such environments. We provide evidence that the currently believed hypothesis that

behaviour-based network scanning detectors (e.g., TRW) exhibit unsatisfactory performance in residential style network traffic [103] is false. We show that this is due to the lack of utilizing information of the characteristics of the monitored environment. In particular, utilizing the monitored network profile to identify benign causes of unsuccessful connection attempts improves significantly the performance of TRW in such environments.

4. **PGRP.** In previous ATT-based login protocols that limit the widespread large-scale online password guessing attacks (e.g., [84, 113]), there exists a security-usability trade-off with respect to the number of free failed login attempts (i.e., attempts triggering no ATTs) versus user login convenience (e.g., fewer ATTs and other requirements). We propose a new Password Guessing Resistant Protocol (*PGRP*) that is more restrictive against attackers than commonly used counter-measures and existing defensive proposals (e.g., [84, 113]) while safely allowing a large number of free failed attempts for legitimate users. Our empirical experiments on two datasets (of one-year duration) gathered from operational network environments show that while PGRP is more effective in preventing password guessing attacks (e.g., PGRP limits the total number of login attempts from unknown remote hosts to as low as a single attempt per username), including attacks empowered by having control of tens or hundreds of thousands of botnet nodes, it also offers more convenient login experience for legitimate users in the sense of requiring answering fewer ATTs for all legitimate users, including those who occasionally require multiple attempts to recall a password.

5. **Reference Baseline of Scanners.** Evaluating the detection accuracy of a network scan detection mechanism on a given network trace requires a reference baseline of identified scanners. We present a novel classification criteria in which remote hosts contacting the monitored network are classified after examining their network traffic over the entire capture period of the trace (as opposed to a short monitoring window in real-time scan detection to make a fast decision). In this classification criteria, each remote host falls into one of six categories

according to a set of benign and scanning behaviour heuristics. We also propose using the exposure maps technique [124] as a proxy for identifying a lower bound on scan detection false positive results.

6. **Evaluation Methodology in the Absence of Absolute Ground Truth.** We are the first, to our knowledge, to explore in detail the problems that can arise when evaluation is based on a GTR rather than AGT. We model the problem of performing evaluation in the absence of AGT, and analyze the requirements of using a GTR for either evaluating one intrusion detector or comparing multiple detectors. We identify the drawbacks of existing approaches for evaluation and comparing network scan detection mechanisms, and discuss challenges specific to scan detection evaluation that are absent in typical intrusion detection systems. We show that a ground truth (i.e., a classification of remote hosts or connection attempts as either scan-related or benign) that is based on a discrete classification criteria could be misleading since the results of typical evaluation metrics will be dependent on the chosen criteria. We present a new evaluation approach for scan detectors designed to address uncertainties in GTR. We believe our evaluation methodology will be of use in evaluating and comparing existing scan detection mechanisms. We also believe that the presented analysis and guidelines for performing an evaluation with real-world network traffic in the absence of absolute ground truth of intrusions apply to broader problems in the network intrusion detection domain.

1.4 Identification of Thesis Work Appearing in Separate Publications

The thesis author is the primary author on the following papers, technical reports, and manuscripts:

1. M. Alsaleh, P.C. van Oorschot. Lightweight Quick and Stateful Network Scanning Detector. In Proc. of the 6th ACM Symposium on Information, Computer and Communications Security (ASIACCS'11), 2011 [11].

2. M. Alsaleh, M. Mannan, P.C. van Oorschot. Revisiting Defenses Against Large-Scale Online Password Guessing Attacks. IEEE Transactions on Dependable and Secure Computing (TDSC). DOI: 10.1109/TDSC.2011.24 [8].

3. M. Alsaleh, P.C. van Oorschot. Revisiting Network Scanning Detection Using Sequential Hypothesis Testing. Journal of Security and Communication Networks (to appear, 2012), John Wiley & Sons. Preliminary version as Technical Report TR-11-08, School of Computer Science, Carleton University, June 2011 [9].

4. M. Alsaleh, P.C. van Oorschot. Evaluation in the Absence of Absolute Ground Truth: Towards Reliable Evaluation Methodology for Scan Detectors. Manuscript, Jun 2011 (submitted to the International Journal of Information Security (IJIS)) [10].

5. M. Alsaleh, D. Whyte, P.C. van Oorschot. A Comparative Evaluation of Behaviour-based Scanning Detection Algorithms Based on the State of Inbound Connections. Manuscript Nov 2011 (manuscript in preparation) [12].

Parts of this additional publication are also presented in this thesis:

6. M. Alsaleh, D. Barrera, P.C. van Oorschot. Improving Security Visualization with Exposure Map Filtering. In Proc. of the 24th Annual Computer Security Applications Conference (ACSAC'08), 2008 [7].

1.5 Organization

The remainder of the thesis is organized as follows. Chapter 2 provides related work and relevant background on network scanning and detection mechanisms. In Chapter 3, we provide an analytical and empirical comparison study of two known scan detection algorithms (TRW and EM) and a novel classification criteria of remote hosts in a given network trace that provides a reference baseline for evaluating the detection accuracy of scan detection algorithms. In Chapter 4, we present and evaluate LQS, a new lightweight, quick and stateful scan detection algorithm. In Chapter 5, we present STRW, a modified algorithm for network scan detection which takes into account causes of TRW false positives, particularly in transient and residential style

network environments. We also propose using the exposure maps technique as a proxy for ground truth for identifying a lower bound on scan detection false positive results. Note that while the chronological order of conception is to first present STRW (Chapter 5) and then LQS (Chapter 4), as STRW is a modified TRW algorithm and as we consider the LQS algorithm a stronger contribution, we instead chose the current order because the evaluation methodology and the two datasets used in Chapters 3 and 4 are similar and are different than those used in Chapter 5. In Chapter 6, we model the problem of performing evaluation in the absence of absolute ground truth, analyze the requirements of using a ground truth reference for either evaluating one intrusion detector or comparing multiple detectors, and present a new evaluation approach for scan detectors designed to address uncertainties in ground truth reference. In Chapter 7, we present PGRP, a new password guessing resistant protocol that significantly improves the security-usability trade-off. We conclude in Chapter 8 by providing a comparative summary of the presented scanning algorithms relative to existing approaches, summarize our main results, revisit the thesis hypotheses, and explore future directions.

Chapter 2

Related Work and Background

We start this chapter by defining terminology and terms commonly used in the field of network scan detection in Section 2.1. An overview of network scanning techniques and algorithms are then given in Section 2.2. Understanding the various techniques of generating network scanning helps in performing a better evaluation of detection mechanisms.

Various approaches for detecting network scanning have been proposed in the literature. The majority of these need to look only at the IP or TCP/UDP packet headers of the network traffic. Different subsets of header contents are analysed by different scan detection mechanisms in order to infer scanning activity. While the analysis usually depends on statistical models, there are machine learning based methods and visual-based mechanisms. Few proposals correlate remote scanners to detect coordinated scans. Related work in these three areas is discussed in Sections 2.3.1, 2.3.2, and 2.3.3, respectively.

In addition, Section 2.4 discusses several scanning worm detection techniques that involve ways to detect automated port scanning (e.g., the rate at which hosts initiate connections to newly visited local hosts) which usually focus on infected machines in the local network. Section 2.4 goes through some other proposals for detecting coordinated scans.

This chapter gives also a detailed background of the TRW (in Section 2.6) and EM (in Section 2.7) scanning detection algorithms which will be studied and evaluated in chapters 3. Background material on the exposure maps (EM) technique is also given in 2.7.

2.1 Network Scanning Terminology and Definitions

Various terminologies have been used in the literature for network scanning. The most commonly used term is *port scanning* which has different definitions. For example, Leckie et al. [58] defined it as "an exercise in intelligence gathering by an attacker" while Jung et al. [45] defined port scanning as "the attacker probes a set of addresses at a site looking for vulnerable servers". While "port scan" is mainly used for single source scanning multiple ports on a single destination, the following are among a few other terms that are used by different researchers:

- Probe: usually used to refer to making a connection attempt from a single source to one port of a single destination to elicit a response in an attempt to discover if the targeted port is offering a service or if the destination host is active.

- Vertical Scanning (Portscan): probing a set of ports on the same IP address to find out the running network services on the IP address or to find out if the corresponding host is active.

- Horizontal Scanning (Portsweep): probing multiple IP addresses for the same port to find out the running network services or to find out active hosts.

- Ping Sweep (ICMP Scan): a single or multiple hosts sending ICMP ECHO requests to a single or multiple IP addresses to find out active hosts.

- Strobe Scan: probing multiple IP addresses on multiple ports to find out the running network services or to find out active hosts.

- Block Scan: probing multiple IP addresses on all ports to find out the running network services or to find out active hosts.

- Coordinated scan (distributed scans): multiple IP addresses probing single or multiple hosts to find out the running network services or to find out active hosts.

For any type of network scanning including the above definitions, a scanner is assumed to lack knowledge fully or partially of the targeted network in terms of knowing what network services are running on the scanned ports, gathering more information about the running services on the scanned ports, or finding out if the contacted hosts are responsive. Thus, we can define a scanner as "a host which initiates a single or multiple connection attempts destined to one or multiple ports in one or multiple destination hosts for the purpose of finding out if all or some of the targeted ports are offering accessible network services or to find out if all or some of the targeted hosts are active".

The above definitions are based on two characteristics of the network traffic:

1. the number of targeted IP addresses (i.e., one, multiple, or all IP addresses in the network),

2. the number of distinct targeted ports (i.e., one or multiple ports). Gates [35, 34] added two other characteristics:

3. the selection algorithm for how an IP address is targeted (i.e., randomly, based on some pattern in the IP addresses, or some contiguous space),

4. the camouflage type that is used to obfuscate the true target (i.e., none, randomly scanning additional IP addresses, scanning IP addresses that meet some property, or scanning all IP addresses in the targeted subnet).

Out of 72 possible permutations of the values of these four characteristics (i.e., $3 \times 2 \times 3 \times 4 = 72$), Gates [34] identified 21 potential adversary types. The set of {IP address, port} pairs the scanner targets in the monitored network is called *scan footprint* [105].

On the other hand, for a scan detector, knowing the objective of a remote host contacting the local network can only be guessed by monitoring the remote host's network traffic. Therefore, reporting a remote host as a scanner requires having convincing evidence(s) of the remote host objective. Given that a fast and correct detection of scanning activity is required for a real-time detector (to enable a better protective responsive before the scanner finds an active vulnerable host), network scanning detection is a challenging problem.

Correlating scanner and scanning detector locations, Whyte et al. [121] characterized network scanning activity in an enterprise network to three types:

- Remote to Local (R2L): the scanner targeting the monitored network is outside the network.
- Local to Local (L2L): the scanner targeting the monitored network is inside the network.
- Local to Remote (L2R): the scanner is inside the monitored network and its scanning target is outside the network.

Network scanning detection techniques could perform differently according to the location of the scanner/detector and the environment conditions of the monitored network. While much of the scan detection literature has focused on R2L scanning activity and assumed an enterprise environment for the monitored network, few proposals studied scan detection on backbone network traffic such as ISPs. The nature of the network environment where the scanning detector is located can be further categorized according to the following factors: number of offered network services, number of servers, number of hosts, IP address range, local host connectivity to the local network and to the Internet, percentage of unavailable packets to the scanning detector (due to using a different routing path, congestion in the network, or the capacity of the detector in processing network traffic) and security policy (in terms of permissible network services and applications on local hosts). For example, an enterprise network with a security policy is often more constrained and stricter than one with an open security policy (e.g., some university environments).

2.2 Network Scanning Techniques

Understanding the various port scanning techniques from basic probes to advanced packet crafting used by advanced hackers is crucial in designing effective scanning detection mechanisms. In this section we review several network scanning techniques for TCP, UDP, ICMP, IP, and ARP protocols.

TCP Scan

There are different ways to utilize the TCP three-way-handshake to check whether the targeted port is running a TCP network service:

1. TCP SYN: a scanner sends a TCP SYN packet (i.e., a TCP packet with only the SYN flag set) to the targeted port. If the targeted port responds with a SYN/ACK packet the port is open (i.e., running a service). The scanner can either send a RST packet or ACK and then RST packets to end the connection. It is also possible to complete the TCP three-way-handshake and then end the TCP connection normally by sending a FIN packet followed by a received FIN/ACK packet (TCP connect scan). Otherwise, if the targeted port did not respond or sent a RST packet then this could be an indication that this port is closed (some firewalls can forge a RST packet to look like it initiated from the target host). However, receiving a RST packet could indicate that the corresponding host is available and responsive.

2. TCP SYN/ACK: a scanner sends a TCP SYN/ACK packet first to the targeted port which falsely indicates that the targeted port has sent an SYN packet earlier. The objective of the scanner is to bypass the blocking rules of stateless firewalls (if any) to find out if the contacted local host is responsive. Receiving back a RST packet indicates that the local host is alive but it does not show whether the port is open or closed. If there is no response received or receiving an ICMP unreachable error both indicate either non-existing host or that the host is behind a firewall. Many routers and firewalls can be configured to block inbound SYN packets (except to public network services like a web server) to prevent external hosts from contacting local hosts or local network services and thus this scanning technique could be effective.

3. TCP ACK: a scanner sends a TCP ACK packet first to the targeted host which falsely indicates that it is an acknowledgement for a data packet sent previously by the local host over an established TCP connection. Similar to the TCP SYN/ACK scan, the objective of the scanner is to bypass the blocking rules of stateless firewalls (if any) to find out if the contacted local host is responsive.

Receiving back a RST packet indicates that the local host is alive but it does not show whether the port is open or closed. However, some operating systems send a RST packet with a non-zero value for the TCP window field if the port is open and zero window field if the port is closed [79].

4. TCP FIN (or FIN, PSH, and URG): a scanner sends a TCP FIN packet first to the targeted port. Here, the lack of a response indicates that the port might be open and a response of a TCP RST packet indicates the port is closed. Receiving no response could also mean that the host is behind a firewall that ignores such packets, while receiving an ICMP unreachable error indicates that this port is blocked by a firewall. Similar response is expected if FIN, PSH, and URG flags are all set or if all flags are unset. This type of response is expected from operating systems that comply with RFC 793 (e.g., Unix-based systems do but Windows systems reply with a TCP RST whether the port is open or close). Other TCP flag permutations could also cause different responses according to the scanned operating system.

5. TCP Idle: by using an intermediate host, an adversary can scan a target without sending any packet with its IP address to the target [92]. First, the scanner sends a SYN/ACK packet to an intermediate host (a zombie) known to be responsive. The scanner will receive a RST packet from this host that contains the current IP ID of this host.[1] Next, the scanner sends a SYN packet to the target host using the intermediate host IP address as the source address. If the targeted port is open, the target host will send a SYN/ACK to the intermediate host which will send a TCP RST packet back to the target host since it actually did not send any packet to the target host. If the port is closed, the target host will send a RST packet to the intermediate host which will ignore this unsolicited RST packet. The adversary will then send another SYN/ACK to the intermediate host to find out how its IP ID has changed. If the intermediate host's IP ID has increased by two (or more) then this host must have sent a packet most likely to the target host which indicates the target port is open.

[1] The IP ID identifies each packet sent by a host and increments usually with one with each packet that is being send, and it can also be used as a fragment identification number.

Otherwise, if the IP ID has increased by only one then the target port might be closed. A significant advantage of TCP idle scanning is its ability to bypass the targeted network firewall by using a responsive local host as the intermediate host. However, if the chosen intermediate host receives many connections (e.g., a busy web server), conclusions drawn from scan responses might not be accurate.

UDP Scan

For scanning a UDP port, a scanner sends an empty UDP packet to the targeted port. If the port is closed, the targeted host is expected to send an ICMP port unreachable packet back to the scanner which also indicates that the host is responsive. Otherwise, if the port is open, most UDP services are expected to ignore the received UDP packet. While this type of scan can bypass firewalls that only filter TCP, it is not effective in finding whether a given port is open or closed. Therefore, a more advanced scanner might send a UDP packet that contains a compatible request to the expected running service in the targeted port hoping to receive a response.

ICMP Scan

This type of scan can only be used to find out if a given IP address is assigned to an active host (i.e., a reachable host through the Internet). An ICMP echo request packet is sent to the target IP address. Receiving back an ICMP echo reply indicates a responsive host. Receiving an ICMP protocol unreachable message indicates that the host is not active or that the probed IP address is not assigned. Most firewalls and routers block such requests by default and thus this technique is not effective.

IP Scan

In an IP Scan, a scanner sends multiple IP packets to a fixed {IP address, port} with the protocol number set in the header to different protocols (e.g., TCP, UDP, IGMP, or IP-in-IP) hoping that the target port will reply to one of these packets. A response using the protocol which was used in the request implies that the corresponding service is active while an ICMP protocol unreachable message (similar to ICMP Scan) signifies that the scanned host is responsive.

ARP Scan

ARP scanning can be used only if the scanner is inside the targeted network to find out if an IP address is assigned to an active local host (i.e., a reachable host through the local network). The Address Resolution Protocol (ARP) is used to determine a network host's MAC address when only its IP address is known. The scanner broadcasts an ARP request packet containing the target IP address. Every host within the broadcast domain receives this ARP request and the device with the specified destination IP address will respond with an ARP reply containing its own MAC address. If no response is received, it is either that the IP address is not assigned to any host within the subnet or that the corresponding host is temporarily unavailable. While local hosts can block IP-based ping packets (i.e., IP scan), they generally can not block ARP requests/responses since otherwise they would not able to operate in the network.

2.3 Scan Detection

Various approaches for detecting network scanning have been proposed. Many of the proposed detection strategies are designed to be deployed at the gateway of an enterprise environment or across multiple detectors covering enterprise subnets. The majority require only IP or TCP/UDP packet headers and aim to identify IP addresses that perform scanning activities. Different subsets of header contents are analysed by different mechanisms to infer scanning activity. While the analysis usually depends on statistical models (see Section 2.3.1), there are machine learning based methods (see Section 2.3.2) and visual-based mechanisms (see Section 2.3.3). In addition, several scanning worm detection techniques involve ways to detect port scanning (e.g., the rate at which hosts initiate connections to newly visited local hosts) focusing on infected machines in the local network (e.g, [104, 95, 118, 44] as discussed in Section 2.4). A few proposals correlate remote scanners to detect coordinated scans (e.g., [129, 35] as discussed in Section 2.5). Some proposals target backbone traffic such as that of ISPs where only uni-directional flows might be available (e.g., [36, 101]), and thus connection state cannot be determined.

2.3.1 Statistical-based Techniques

Enterprise Environment

The most popular analysis methods for scanning detection are statistical-based methods and in particular threshold-based. Network Security Monitor (NSM) IDS [40] is one of the early approaches which only looks at the destination IP addresses contacted by a remote host. NSM considers a remote host anomalous once it contacts more than 15 local hosts within a time window (the precise duration of which was not specified), or when the remote host attempts a connection to a non-existing host.

Kato et al. [49] proposed a real-time IDS for detecting network attacks that looks for TCP ACK/RST packets as a sign of scanning. They set a threshold on the number of TCP ACK/RST returned to the same remote host within a specific time window. The remote host is labeled as a scanner once the threshold is exceeded. A probabilistic model used by Leckie and Kotagiri [58] considers both the number of local hosts or ports accessed by a remote host, 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 host being benign or a scanner. Such an approach obviously requires rich knowledge of the monitored network and dynamic updates of the host/port probabilities according to the network traffic and architecture changes.

Spice [105] is a port scan detector designed to detect stealthy scans using a statistical model to rank incoming traffic according to the targeted destination IP address and port. Packets sent to rarely accessed IP address/port combinations are considered more anomalous and thus the rank of these packet is expected to exceed an adjustable anomaly score threshold where an alert is generated. Using a correlation graph, the generated events are further analysed by a correlation engine to identify distributed scans.

The system introduced by Robertson et al. [89] scores each remote host based on the number of unique destination IP/port pairs of failed connection attempts; 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. [50] calculate a normal distribution of destination IP addresses/ports in a network and then use some statistical tests on the network traffic to declare port scans.

A scan detection preprocessor plug-in called sfPortscan [90] in Snort [91] generates an alert when a remote host attempts to connect (using either TCP or UDP) to more than a predefined threshold of local hosts (4 IP addresses is the default threshold) or to more than a predefined threshold of ports (19 is the default threshold value for the number of contacted ports either in one or multiple hosts) within a predefined time window (1 minute is the default value). The plugin scan detection method is similar to the tools developed by Fullmer et al. [32] and Navarro et al. [76]. By not exceeding the probing threshold within the specified time window, once these parameters are known, an adversary can easily evade detection of this Snort plug-in.

Another threshold-based approach is the one adopted by early versions of the Bro intrusion detection system [2] where a remote host is considered a scanner if it contacts more than a predefined threshold of local hosts or by contacting too many ports. Newer releases of Bro [2] (starting from v0.4) come with a scan analyzer that considers a large number of tunable parameters to adjust the scan detection method (e.g., whitelisting ports, enabling ICMP scan detection, and enabling detection of scanning activity from local hosts). By default, Bro's scan analyzer skips some services which are known to regularly receive excessive background noise and has some rules to ignore what looks like backscatter traffic [80].

Jung et al. [45] proposed TRW (Threshold Random Walk), a fast online scan detection algorithm. TRW classifies each remote host contacting a target network as either benign, scanner, or pending according to the ratio of the remote host's successful or failed connection attempts in the inbound network traffic within a time window. By conducting experiments on network traces from two sites and by using some empirically determined parameter settings (based on some particular datasets), the authors found that for that specific environment of the studied datasets, TRW works well when parametrized such that the resulting threshold is 4 consecutive failed connection attempts destined to 4 different local hosts for a remote to be classified as a scanner. TRW is also implemented as a Bro policy[2] such that scanners' traffic can be dropped by setting the appropriate interface between Bro and the corresponding network router. The TRW algorithm is explained in greater detail in Section 2.6.

[2]A Bro policy is a script written in the Bro language. Using Bro network activity events, the policy usually analyzes the network activity and initiates actions accordingly.

A simplified variant of TRW that requires fewer system resources (memory footprint in particular) and can detect vertical scanning (i.e., a port scan of more than one port on a single host) is proposed by Weaver et al. [118]. The authors also proposed a suppression algorithm for worm containment with dynamically adjustable threshold. A similar modification to TRW by Nagaonkar[72] considered vertical scanning and extended TRW to detect UDP and ICMP scans. Nagaonkar used a Bloom filter (a probabilistic space-efficient data structure for testing set membership) to filter the input to TRW so that only unique {remote IP address, destination IP address, destination port, protocol} quadruple are processed by TRW. The idea is that this eliminates the need for keeping the state of previous inbound connection attempts between remote and local hosts. In addition, they increased the TRW time window of keeping the state of remote hosts' connection attempts to detect stealthy scanners.

Using a set of statistical features, Simon et al. [97] proposed a data mining classifier to detect scanners avoiding P2P and backscatter traffic. Their main rule in differentiating between scanning and P2P traffic relies on their observation that unlike scanners, the probability that remote hosts initiating P2P traffic make connection attempts to different local hosts on the same closed destination port is negligible. While P2P traffic that does not follow this rule might be mistakenly flagged as scanning activity, it is also easy for scanners to evade detection by contacting different ports at different local hosts so they do not exceed the corresponding thresholds.

Testing TRW on two 3G cellular network datasets, Falletta and Ricciato [28] observed that TRW marks remote hosts with P2P activity as scanners. As an alternative to TRW, the authors proposed a heuristic metric (without apparent justification) to differentiate between scanning and P2P activity.

Generally, given that the main dilemma seems to be that a lower false positive rate usually leads to a lower detection rate, we believe that it is hard to find the right tradeoff between detection (i.e., true positive rate) and false positive rate in such threshold-based approaches. In addition, such approaches require a priori knowledge of the monitored network.

Backbone Environment

While the above scan detection mechanisms target mainly enterprise environments, a few other approaches in the literature to date have focused on backbone traffic such as ISPs where usually only uni-directional flows are available (due to lack of accurate connection state). Gates et al. [36] used a Bayesian approach with logistic regression modeling to choose weights for a set of six selected metrics which they found could indicate the presence of scanning activity in TCP traffic (e.g., the ratio of flows with no ACK bit set, to all flows). Two metrics [36] have positive weights so that as their values increase, the likelihood that the event includes scanning increases: the ratio of flows with no ACK bit set to all flows and the ratio of the number of unique destination IP addresses to the total number of flows. The other four metrics have negative weights: the average number of source ports per destination IP address, the ratio of flows with fewer than three packets to all flows, the ratio of flows with a backscatter flag combination to all flows, and the ratio of flows with an average of more than 59 bytes per packet to all flows. This scan detection method is implemented and used in the SiLK toolset [21]. In addition to expert opinions, the selection of these metrics is also based on the dataset they have studied which is a main limitation of this model. Moreover, the approach requirement to construct each event in terms of time and number of flows makes this technique impractical for immediate detection of scanners. Rather, the technique is more suitable for scan detection that are used for forensic analysis or that does not require immediate response.

Sridharan et al. [102] adapted TRW to fit the backbone environment so that flows having more than one packet and single packet flows without a SYN flag are defined as successful connections whereas single packet flows with the packet's SYN flag set are defined as failed connections. Their proposed algorithm, TAPS, does not require bi-directional flows (i.e., connection state might not be known) or a priori knowledge about the configuration of the monitored network. For practical implementation of TAPS, Sridharan and Ye [100, 101] used probabilistic counters from the domain of data streaming and queueing theoretic analysis to bound the buffer size. The resulting algorithm flags a remote host as a scanner once the ratio of the number of destination IP addresses the remote host contacted over the number of destination ports the

remote host attempted exceeds a certain threshold within a time window. Since the technique assumes that a scanner is expected to go to a large number of destination IP's over a small range of ports and vice versa for benign sources, it is easy to evade detection by simply not exceeding the chosen threshold, once it is known.

2.3.2 Machine Learning based Techniques

To detect slow, stealthy probe and DoS attacks, Basu et al. [14] proposed a system based on a neural network classifier (a multi-layer perceptron). In addition to using some common packet header features, they extracted some traffic statistics features based on a time window to detect anomalies. Further improvements to the system have been proposed by Streilein et al. [110]. Similar to other network profiling approaches, there is always a need to update the network profile in the detection system to reflect network configuration changes.

Simon et al. [97] developed a data mining classifier using a set of features that encode expert knowledge about scan detection. The features include statistics aggregated over IP addresses/ports of the monitored network. Although they showed comparable results to the TRW technique, it is not clear how their model heuristics will perform on datasets from different networks and how to adjust these heuristics accordingly.

2.3.3 Visual-based Techniques

Visual-based scan detection techniques depend on showing graphical representations of selected data dimensions of network traffic so that network security analysts can identify scan activity patterns. The scan detection scheme in GrIDS [107] is an early graphical approach that shows hosts activities and connectivity over time where a graph of one host contacting many local hosts could indicate a possible scan activity. In addition to scalability and configurability issues, the proposed system can only be used for detecting large-scale attacks and not for stealthy activities.

The Spinning Cube of Potential Doom [57] is an animated 3D visual display of network traffic to monitor network links, and enable searching for traffic that potentially violates access and usage policies. Each axis represents a different component

of a TCP flow header (the typical setting is X for the local IP addresses, Y for the destination port numbers, and Z for the remote IP addresses). While this tool shows clearly the source of horizontal and vertical scans, stealthy scan activity looks similar to normal traffic. Moreover, although the user can spin the 3D cube, the 3D view suffers from occlusion and disorientation (i.e., determining the exact coordinates of values in the graph).

Lakkaraju et al. [55] introduced the NVisionIP tool to visually represent network flows. Among other security-related events the tool helps in identifying, obvious network scanning patterns can be discovered by looking at a 2D diagram in which remote hosts are laid out in the monitored network subnets (subnets in the x axes and remote hosts in the y axes, where the assumption is that the entire monitored class-B address space is unlikely to be populated with hosts). Conti et al. [24] presented several visualization techniques to identify network attacks including a parallel coordinate visualization to detect attacks. The parallel coordinate plot used source/destination IP addresses and source/destination ports as axes. This visualization can only be helpful to identify obvious scanning patterns (e.g., horizontal and vertical scans), and it is hard to identify scanning patterns with high volume of network activity.

Muelder et al. [18] proposed using PortVis, a network tool to visualize activity on each TCP port by choosing axes that correspond to TCP header fields (the tool was first introduced by McPherson et al. [66] for detecting network security events), for scan detection. Based on the chosen axes, the tool shows a grid and fills each cell of the grid with a color that represents the network activity. Three main views are available: the whole traffic at low resolution, all ports at one point in time, and a fine-grained detailed view that focuses on an individual port activity. In addition to noticing horizontal and vertical scans in the second view, setting the first view as the ratio of destination addresses to source addresses helps in finding other scan patterns. The first view can also represent the difference between the number of sessions and the number of source addresses which helps filtering out vertical scans and revealing horizontal scans. Stealthy scan activity from the same source and horizontal scans that occur on ports that are commonly used are both difficult to detect using this tool.

Irwin et al. [41] used the InetVis [114] network traffic visualization tool for detecting network scanning. Using network telescope filtered traffic (i.e., inbound traffic targeting the dark address-space of the network), scan events of interest are identified and then evaluated by comparing the results with those generated from Snort [91] and Bro [82] scan detection scripts.

Generally speaking, network scanning visualization tools require continuous monitoring by security analysts of visualizations that represent recent data. With high volume of network traffic and limited resources in terms of security analysts, it seems highly impractical to detect scanning activity on the fly. Instead, scanning visualization tools are more appropriate for forensic uses as opposed to immediate detection and response. Moreover, the identification of scanning activity relies on manual inspection, that is, analysis of some visualizations by a security analyst to find out scanning activity. While security analysts might discover new scanning patterns, some clear patterns might unwittingly get through undetected. Scalability, in terms of size of network and volume of traffic, is another common drawback among many network traffic visualization tools. Proposed visual representations usually can be helpful using only small-size network traces but could suffer from occlusion and disorientation as soon as more network traffic get visualized. In addition, stealthy scans require monitoring the traffic over a long period of time which might not be possible in many visualization techniques.

2.4 Scanning Worm Detection

A *network worm* is a malicious software program that can replicate itself and infect other hosts through the Internet or the local network where the infected machine resides. Network scanning is a common and effective reconnaissance technique for worms to find vulnerable hosts to infect (such as Code Red, Slammer, or Blaster worms). In addition to random scanning of the IP address space, some worms use the infected host machine to scan all the hosts in its subnet. More advanced worm scanning techniques include hit-list scanning,[3] permutation scanning,[4] and topological

[3] A previously generated list of existing hosts which are possibly vulnerable.

[4] All worms share a common pseudo random permutation of the IP address space. If a worm detects that a host has been already infected, it chooses a new random start point in the permutation.

scanning[5] [106].

On the other hand, there are several scanning worm detection techniques that involve ways to detect port scanning focusing on infected machines in the local network. A popular technique is detecting the rate at which remote hosts initiate connections to newly visited local hosts, as worms tend to make many connection attempts to many local hosts in a very short time [125].

Staniford [104] studied containment of random scanning worms on enterprise environments and showed how to limit the number of first-contact connection attempts that a local host initiates to a given destination port to a predefined threshold. The conducted experiments in this study showed that a threshold of 10 is small enough to limit moderate to fast worms and large enough to not disturb normal traffic. However, a single host can easily exceed this threshold through web browsing since visiting a web page usually involves loading web content from numerous sites in a very short time. The author recommended dividing the enterprise network into cells that can be adjusted to keep the number of vulnerabilities per cell roughly constant to improve worm containment.

A hybrid approach that combines a variation of TRW and a credit-based connection rate limiting algorithm is proposed by Schechter et al. [95] to detect fast scanning worms. The aim is to detect worms that can generate thousands of connection requests before being caught if only TRW is deployed for scan detection. They proposed modifying TRW so that it evaluates connection events in reverse chronological order such that the most recent observations are processed first in calculating the likelihood ratio of the host. This way TRW makes a faster decision in flagging local hosts transited from a benign state to an infected state. In addition, for connection rate limiting, each local host is given an initial balance of 10 credits. Whenever a first contact connection request from the local host is observed, a credit is subtracted from the sending host's balance. If the successful acknowledgment of a first-contact connection is observed, the host that initiated the request is given two credits (in addition to the current balance). First-contact connection requests are blocked if the host balance is zero. In addition to the drawback that limiting the rate at which

[5]The worm utilizes information contained on the infected host to select new target hosts.

first-contact connections can be initiated could block some legitimate hosts (e.g., because of popular web mashups [71], local hosts accessing some of these pages might be blocked), this approach is unable to detect stealthy probes.

Whyte et al. [121] proposed monitoring DNS requests from local hosts so that if there is no DNS request from the local host before a new connection is initiated, the connection attempt is considered anomalous and it is considered an indication of a possible worm infection. Network services that might not require prior DNS queries are whitelisted. The technique can be used to detect scanning worm propagation within a local network and from the local network to the Internet. However, this technique can not be used to detect scanning worm propagation from Internet to the local network.

Whyte et al. [122] also presented a technique to detect worm propagation within enterprise network cells. An anomaly score is assigned to each local host according to the following infection indicators: connection attempts to IP addresses outside the set of internal IP addresses a host normally interacts with, increases in the average number of ARP (Address Resolution Protocol) requests a local host issues per unit time, and connection attempts to unused internal IP addresses (i.e., dark address-space of the local network). Local hosts in the same network cell use ARP to map an IP address to the corresponding physical MAC address. A worm scanning other local hosts within the infected host's subnet is expected to send many ARP requests to resolve other local IP addresses for the first time. Moreover, ARP requests might be generated for nonexistent IP addresses. In this proposal, the number of ARP requests is recorded for each local host over discrete sample intervals during a training period. The anomaly score of a local host is set to the number of ARP requests that are more than the summation of the mean and double the standard deviation of the number of requests during the training period. An alert is generated when the summation of the three infection indicators' anomaly scores exceeds a predefined threshold.

A simplified variant of TRW that requires fewer system resources (memory footprint in particular) and can detect vertical scanning (i.e., a port scan of more than one port on a single host) is proposed by Weaver et al. [118] to detect TCP and UDP scanning worms. They used an approximate cache data structure to find out

whether a connection has been established for both inbound and outbound connection attempts (in addition to source and destination IP addresses, source and destination ports are also stored). Approximate caches are usually used when the data that needs to be stored is larger than the available memory so that two elements can map to the same location in the cache (a Bloom filter is one type of approximation cache). The updated version of TRW classifies remote hosts only as scanner or unknown and not as scanner, benign, or unknown as in the original TRW. The authors argue that their simplifications of the TRW algorithm might increase the false negative rate but not the false positive rate. The authors also proposed a suppression algorithm for worm containment with dynamically adjustable threshold. Several possible attacks are discussed including exploiting the approximation caches' hash and permutation functions.

To address the drawbacks of the Schechter et al. proposal [95], Jung et al. [44] proposed an algorithm for detecting fast-propagating worms that use high-quality targeting information (e.g. worms that spread in a topological fashion) by combining the TRW algorithm and a new algorithm (which they called rate-based sequential hypothesis testing or RBS) that analyzes the rate at which hosts initiate connections to new destinations (i.e., fan-out rate). The RBS algorithm depends on sequential hypothesis testing to adapt its decision in response to the available measurements which reduces both false positives and false negatives. The proposed algorithm, however, is expected to generate high false positive rate when some applications that resemble worm network activity in terms of fan-out rate and high number of failed connection attempts (e.g., P2P) are running in the monitored network. Furthermore, stealthy scanning worms with good quality hit lists might not be detected. The algorithm also requires large memory space per local host.

2.5 Correlating Scan Sources

Streilein et al. [110] presented some improvements to a previous IDS (based on multiple neural network classifiers) to detect stealthy and distributed probes. They maintained and analyzed probe events information (source/destination IP network, time and duration of the probes) to group distributed scans.

Robertson et al. [89] introduced a surveillance detection system using cascading filter design that coordinates a series of heuristics across extrapolated connection records, individual probes, scans and coordinated scanning groups. The authors claimed that their system can detect scans and probes in high-bandwidth environments with low false positive rate. Their system can detect distributed scans from sources within the same network subnet.

Yegneswaran et al. [130] presented an empirical analysis of Internet intrusion activity using a large set of network intrusion detection systems (NIDSs) and firewall logs collected over a four month period. For the datasets used, they found that scanning events' source IP addresses are widely disbursed, the distribution of attempts per source IP follows a power law, and the distribution of source IP addresses of the non-worm intrusions as a function of the number of attempts follows Zipfs law. Their analysis shows that a large proportion of the daily scans are coordinated or come from distributed sources.

Muelder et al. [70] presented a visualization methodology for characterizing network scans. Variations in arrival time of the scanning connections help in analyzing some patterns that can be found in the timing of network scans. In the proposed visualization each node represents a scan and an edge between any two nodes is weighted according to the wavelet statistical comparison [37] between the two nodes where nodes with higher match percentages are placed closer to each other. Any given node can be colored according to destination port, mean scan rate, elapsed time, or number of connection attempts. Clusters of nodes in the graph represent similar scan patterns and thus scans in one cluster indicate a possible correlation between their source IP addresses. Another detailed view shows the third byte of the destination IP address as the X axis and the fourth byte of the destination IP as a the Y axis. Validation of the effectiveness of this visualization in terms of true relations between nodes in each cluster was not provided. A follow up work by Muelder et al. [69] explored using associative memory learning techniques (in particular, BAM machine learning algorithm[6]) to directly compare network scans in order to better characterize the sources

[6]The bidirectional associative memory (BAM) [53] is the minimal two-layer nonlinear feedback neural network. Bidirectionality is introduced in neural nets to produce two-way associative search for stored associations.

of these scans. The classification results are then combined and compared with the previous visualization.

Stockinger et al. [108] presented a set of parallel algorithms that demonstrate how bitmap indexing (an efficient selection mechanism) speeds up computing conditional histograms on very large datasets. They then used parallel algorithms to visually represent conditional histograms for detecting distributed scans (using simple 2D and 3D histogram visualizations).

Gates et al. [35] provided an algorithm to detect coordinated horizontal and strobe scans against contiguous address spaces. The proposed detection algorithm is based on solutions to the set covering problem. To find coordinated scans, scanning events are combined such that a large portion of the information space is covered with minimal overlap. The author also defined a measure of the number of false positive coordinated scans detected and found that her approach has an average of one false coordinated scan per dataset.

2.6 Background on TRW

This section reviews the TRW approach and design and gives pseudo-code of the TRW algorithm for later reference.

2.6.1 TRW Approach and Design

TRW is a well known and fast online scan detection algorithm [45]. The algorithm aims to classify and label remote hosts contacting a network as either being benign or a scanner. The possible response actions upon classification are beyond the scope of this algorithm. The distinction criterion is simply based on a ratio determined by the probability a benign remote initiates a successful connection, the probability that a scanner initiates a successful connection, and the number of successful and failed connections attempts (with a time window) initiated by the remote host towards newly visited local addresses in the monitored network. That is, the connection likelihood ratio Λ of each remote host will be updated only when the remote host attempts a connection (whether successful or unsuccessful) with a local host for the first time, given that the remote has not been classified as being either benign or

a scanner (i.e., it is in a pending state). Subsequent connection attempts from the remote host to the same local host will not be considered when updating the ratio (whether to the same destination port or to different ones).

The reason behind such a design is the significant disparity the authors observed [45] (through their empirical study on network traces gathered from different networks) between the frequency with which connections to newly visited local addresses are successful for benign hosts versus scanning hosts. As scanners usually seek to discover the available services in the target network, scanners are more likely than legitimate remote hosts to attempt connections to hosts which either do not exist or do not offer the requested service. This characteristic of scanning systems occurs as the total number of possible offered services is much greater than the total actual number of services offered in a network.

Since observations (i.e., connection attempts from a remote host to newly visited local addresses) arrive sequentially rather than all at once, TRW classification decisions are based on sequential hypothesis testing where there are two competing hypotheses: the remote host is benign or it is a scanner. Binary hypothesis tests for sequential observations make decisions as the data arrives according to the new value of the sequential *likelihood ratio* (the updated value of the connection ratio in the case of TRW). For each new event, the updated value of the likelihood ratio is compared to an upper and lower threshold. If the ratio of a remote host becomes less than or equal to the lower threshold (η_0 in Figure 2.1), the hypothesis that the remote host is benign is accepted. Likewise, if the ratio becomes greater than or equal to the upper threshold (η_1 in Figure 2.1), the hypothesis that the remote host is a scanner is accepted.

2.6.2 TRW Algorithm

TRW tests two hypotheses: H_0, the hypothesis that a given remote host r is benign and H_1, the hypothesis that r is a scanner. Let Y_i be a random variable that represents the outcome of the first connection attempt made by the remote source r to the i^{th}

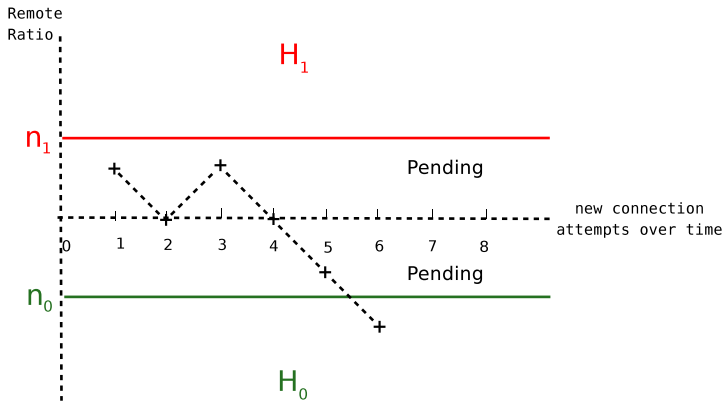


Figure 2.1: Using the likelihood ratio to classify remote hosts.

distinct local host, where:

$$Y_i = \begin{cases} 0 & \text{if the connection attempt is a success} \\ 1 & \text{if the connection attempt is a failure} \end{cases} \tag{2.1}$$

Assuming that $Y_i|H_j$ for $i = 1, 2, ...$ are independent and identically distributed, we can write the following probabilities of the Bernoulli random variable Y_i:

$$\Pr[Y_i = 0|H_0] = \theta_0 \qquad \Pr[Y_i = 1|H_0] = 1 - \theta_0 \tag{2.2a}$$

$$\Pr[Y_i = 0|H_1] = \theta_1 \qquad \Pr[Y_i = 1|H_1] = 1 - \theta_1 \tag{2.2b}$$

The algorithm assumes that $\theta_0 > \theta_1$ which means that the probability of a successful connection is supposed to be higher from a benign remote host than from a scanner. Starting from an initial value of 1, the likelihood ratio Λ for a remote host r is updated whenever a new event i is observed (a new event occurs when r attempts to connect to a new local host) as follows:

$$\Lambda(r) = \Lambda(r) \; . \; \frac{\Pr[Y_i|H_1]}{\Pr[Y_i|H_0]} \tag{2.3}$$

where:

$$\frac{\Pr[Y_i|H_1]}{\Pr[Y_i|H_0]} = \begin{cases} \frac{\theta_1}{\theta_0} & \text{if the connection attempt is a success} \\ \frac{1-\theta_1}{1-\theta_0} & \text{if the connection attempt is a failure} \end{cases} \tag{2.4}$$

After updating the likelihood ratio Λ for a remote host r, the new ratio value is compared to a lower threshold η_0 and an upper threshold η_1 as follows:

$$\begin{aligned} \eta_0 < \Lambda(r) < \eta_1 &: \text{ no hypothesis is chosen (i.e., the state of } r \text{ is pending)} \\ \Lambda(r) \leq \eta_0 &: \text{ hypothesis } H_0 \text{ is chosen (i.e., } r \text{ is declared benign)} \\ \Lambda(r) \geq \eta_1 &: \text{ hypothesis } H_1 \text{ is chosen (i.e., } r \text{ is declared a scanner)} \end{aligned}$$

The pending state means that there is not enough evidence to determine if r is either benign or a scanner and thus more events are required in order to choose a hypothesis. The hypothesis chosen by the algorithm can be either:

Correct: a *true positive* (i.e., detection) occurs when hypothesis H_1 is chosen while r is a scanner, or *true negative* (i.e., nominal) when hypothesis H_0 is chosen while r is a benign host.

Incorrect: a *false positive* occurs when hypothesis H_1 is chosen while r is a benign host or *false negative* when hypothesis H_0 is chosen while r is a scanner.

In sequential hypothesis testing, the lower threshold η_0 and the upper threshold η_1 can be bounded by both the probability of detection variable P_D, and the false positive probability P_F. Having a set of sequential observations $Y_1, ..., Y_n$, and assuming that at the n^{th} observation the upper threshold η_1 is reached and thus hypothesis H_1 is chosen, implies that the likelihood ratio Λ is greater than or equal to η_1:

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

The probability of all sample paths where H_1 is selected when H_1 is true (i.e., $\Pr[Y_1, ...Y_n|H_1]$) and the probability of all sample paths where H_1 is selected when

H_0 is true (i.e., $\Pr[Y_1, ...Y_n|H_0]$) can be replaced by P_D and P_F respectively:

$$\frac{P_D}{P_F} \geq \eta_1 \tag{2.6}$$

Likewise, we can deduce:

$$\frac{1 - P_D}{1 - P_F} \geq \eta_0 \tag{2.7}$$

If the desired detection probability β and false positive probability α can be set (ideally having $\beta \leq P_D$ and $\alpha \geq P_F$), as explained by Jung et al. [45], the upper threshold η_1 and the lower threshold η_0 can be simply chosen as follows:

$$\eta_1 = \frac{\beta}{\alpha} \quad \text{and} \quad \eta_0 = \frac{1 - \beta}{1 - \alpha} \tag{2.8}$$

The classification of a remote host by TRW can be kept for a specific time window after which the remote host state can be set back to pending for further evaluation. The network security administrator is expected to set these four parameters (θ_0, θ_1, α, β), otherwise the default values are used.

Algorithm 1 gives pseudo-code of the Bro 1.4 implementation of TRW core algorithm [2, 82]. A discussion about selecting the right parameters for TRW is presented in section 4.2. The keyword "def" denotes the default parameter value. Each element in the sets S, B, FC, SC, R, and the table L has a "write-expiry" interval such that each element is removed when the given period of time (I_1, I_2, I_3, I_4, I_5, or I_6) has lapsed since the last time the element was inserted in the set/table.

The function `FailedConn(conn)`[7] returns true when the connection attempt `conn` is unsuccessful, whereas the function `SuccessfulConn(conn)`[8] returns true when the connection `conn` is successful.

[7] FailedConn(conn): either (a) RST packet is sent by destination after receiving SYN packet from source, or (b) SYN packet or midstream traffic is sent by the source but no SYN-ACK packet is sent by the destination for at least 5 minutes.

[8] SuccessfulConn(conn): SYN-ACK packet is sent by the destination.

Algorithm 1: TRW (returns True when a new IP is classified as a scanner).

INPUT:
β(def:=0.99), α(def:=0.01), θ_0(def:=0.8), θ_1(def:=0.2)
I_1(def:=1hr), I_2(def:=1hr), I_3(def:=0.5hr), I_4(def:=0.5hr), I_5(def:=0.5hr), I_6(def:=0.5hr)
C `//data structure holding current connection information.`

OUTPUT:
S (def:=$\emptyset$, expires after I_1) `//set of scanners' IP addresses.`
B (def:=$\emptyset$, expires after I_2) `//set of benign IP addresses.`
FC (def:=$\emptyset$, expires after I_3) `//set of IP address pairs with failed connection.`
SC (def:=$\emptyset$, expires after I_4) `//set of IP address pairs with successful connection.`
R (def:=$\emptyset$, expires after I_5) `//set of friendly remote IP addresses.`
L (expires after I_6) `//table of likelihood ratios of remote hosts (Λ).`

```
1  begin
2    if IsLocalAddress(C.srcIP) then
3      if C.dstIP ∉ S then
4        add C.dstIP to R
5      end
6      return (False) //since it is outbound connection
7    end
8    if C.srcIP ∈ (S ∪ B ∪ R) then
9      return (False) //remote is already flagged as scanner, benign, or friendly.
10   end
11   if FailedConn(C) ∧ ([C.srcIP, C.dstIP] ∉ FC) then          //see Note [7]
12     add [C.srcIP, C.dstIP] to FC
13     ratio ⇐ (1 − θ1)/(1 − θ0)
14   else if SuccessfulConn(C) ∧ ([C.srcIP, C.dstIP] ∉ SC) then  //see Note [8]
15     add [C.srcIP, C.dstIP] to SC
16     ratio ⇐ θ1/θ0
17   else return (False)
18   if (an entry in L already exists for C.srcIP) then
19     L[C.srcIP] ⇐ L[C.srcIP] ∗ ratio
20   else
21     add new entry for index C.srcIP into L
22     L[C.srcIP] ⇐ ratio
23   end
24   if L[C.srcIP] > (β/α) then
25     add C.srcIP to S
26     return (True)
27   else if L[C.srcIP] < ((1 − β)/(1 − α)) then
28     add C.srcIP to B
29   end
30 end
```

2.7 Background on EM

This section reviews Exposure Maps EM approach and design and gives pseudo-code of the EM algorithm. It then gives an overview of how the EM algorithm can be

used to effectively reduce the volume of network traffic for the security visualization techniques, focusing on possible malicious behavior.

2.7.1 EM Approach and Design

Exposure maps [123, 120] is a table of the services offered by a particular network. This table can be built by an automated tool based on monitoring how internal hosts respond to incoming connection attempts. Host exposure map (HEM) is constructed by passively observing a local host networks traffic. A HEM tuple consists of the internal host IP address, open port number and the corresponding protocol. Network exposure map (NEM) is the union of HEMs (i.e., by recording all {IP address, port, protocol} tuples which respond to outside connection attempts).

To construct the NEM table, all outgoing TCP flows containing SYN-ACK flags are observed and recorded, with every local host that was seen responding with SYN-ACK flags added to the NEM. For UDP traffic, UDP ports in the NEM are added when 2 hosts communicating with the same source and destination port pairs (Host1 using a as a source port and b as a destination port, and Host2 using b as source and a as destination) are tracked within a small time period. The NEM is built over a reasonable training period to observe local host responses.

Finally, the NEM is *vetted* where the offered services, as indicated by the NEM, are confirmed to be allowed by the network security policy. Ideally in the exposure maps technique the training period should be long enough to include legitimate traffic going to all open ports on the network in order to populate the NEM. Probes to closed ports during this training period will not establish sessions and therefore will not add entries to the NEM. Thus, the training period does not need to be free from probes.

In case there is no uniform network security policy in the monitored network, the NEM table construction does not require a training period. Instead, the *unvetted* NEM table will always be updated whenever a new service is offered in the monitored network [120]. Exposure maps can be used for different network applications. Among others, network scanning detection is a primary application of exposure maps technique.

Algorithm 2: EM (returns True when a new IP is classified as a scanner after the training period).

INPUT:
C `//table of current connection information`
TrainingDate `//the end date/time of the training period.`

OUTPUT:
NEM (global variable, def=∅) `//set of offered services in the network {IP, port}.`
S (global variable, def=∅) `//set of scanners' IP addresses.`

```
1  begin
2      if IsLocalAddress(C.srcIP) then
3          return () //Since it is outbound connection
4      end
5      if SuccessfulConn(C) then                                    //see Note [8]
6          if CurrentDate() <= TrainingDate then
7              add [C.dstIP, C.dstPORT] to NEM
8          end
9          return (False)
10     else if ([C.dstIP, C.dstPORT] ∉ NEM) ∧ (CurrentDate() > TrainingDate) then
11         add C.srcIP to S
12         return (True)
13     else
14         return (False)
15     end
16 end
```

2.7.2 EM Algorithm

Detecting scanning activities using exposure maps is a straightforward process. For a vetted NEM, each new incoming connection attempt is checked to determine if it matches an existing entry. If it does, it is considered legitimate traffic, otherwise it is considered a scan event. For an unvetted NEM, each new incoming connection attempt is checked against the current NEM table. If the connection matches an entry in the NEM, it is labeled as legitimate traffic. Otherwise, the algorithm waits until it is able to determine if the connection is a success or failure. If the connection fails, it is labeled as an atomic scan event and remote host IP address is added to the scanners list. Otherwise, a new entry is added to the NEM of the newly observed service. Algorithm 2 gives pseudo-code of the EM algorithm.

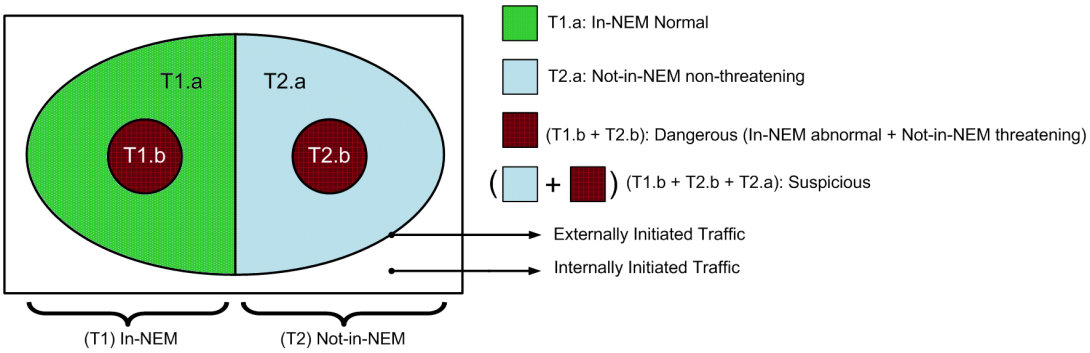


Figure 2.2: Graphical representation of filtered flow subsets [7].

2.7.3 Security Visualization Filtering with EM[7]

EM can be used to filter raw network data [7], in order to focus visualization on network traffic from remote hosts that exhibit scanning behaviour and have contacted some responsive services in the target network. This helps in reducing the network traffic for the visualization process, resulting in visible patterns and insights not previously apparent. Using EM, inbound connection attempts can be categorized into a number of disjoint sets (see Fig. 2.2), in logical tables with semantics as follows.

- **Table T1: In-NEM.** This table contains flows (i.e., connection attempts) destined to a host/port combination offering an authorized service (i.e., to an authorized open port in the local network). This table is also logically partitioned into two sub-tables.
  - ✦ **T1.a: In-NEM normal.** This table contains flows that are considered ordinary, since their source IP addresses have only attempted connections to authorized services offered by the network in question (i.e., destined to an authorized open port).
  - ✦ **T1.b: In-NEM abnormal.** This table contains flows initiated by source IP addresses that also have flows in T2. These flows are labeled 'suspicious' because normally, a host does not attempt connections to closed ports while also accessing legitimate services.
- **Table T2: Not-in-NEM.** This table contains flows destined to a host/port

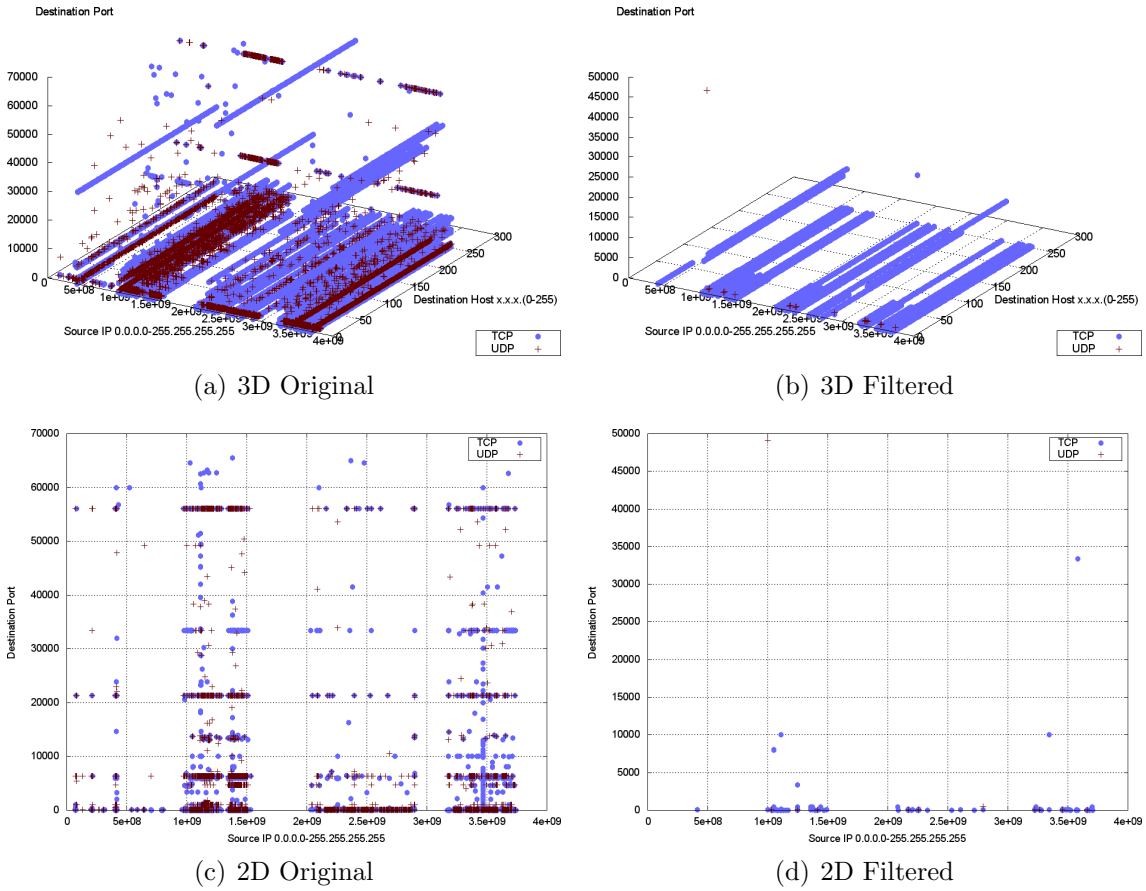
[7]This section includes joint work [7] with Ph.D. student David Barrera.

combination for which no authorized service is offered (i.e., closed port). It is logically partitioned into two sub-tables.

- ✦ **T2.a: Not-in-NEM non-threatening.** This table contains flows in T2 and whose source IP addresses have no flows in T1. Exposure map filtering assumes these connection attempts are not a significant threat to the target network since sources, all of whose probes have been to closed ports, have not learned what is considered significant information from the target network (i.e., have not learned what services are offered).
- ✦ **T2.b: Not-in-NEM threatening.** This table contains flows in T2 and whose source IP addresses also have flows in T1. Thus, the source IP address of these flows have queried both legitimate offered services and closed ports.

- **Table T3: Suspicious.** This table includes all flows in T2 (T2.a and T2.b) plus T1.b. this is called 'suspicious traffic' because these source IP addresses have probed at least one closed port in the network.
- **Table T4: Dangerous.** This table includes all flows in T1.b plus T2.b. This represents traffic from IP sources that probed at least one closed port and also attempted to connect to an open port. According to the philosophy motivating the exposure maps technique, these are more likely to represent malicious flows since these IP sources, if adversaries might attempt to send exploits to the open ports that they have discovered.

For example, for the dataset described in [7], Figure 2.3 graphs the full source IP address (plotted as an integer from 0 to approximately 4.2 billion), the target destination host, and the destination port. Item (a) in each graph pair represents the visualization of the unfiltered network traffic, while the filtered visualization is item (b) in each graph pair. In the unfiltered visualization (item (a) in each graph pair), all inbound connection attempts are plotted. In the filtered visualization (item (b) in each graph pair), inbound connection attempts initiated from only the *dangerous table* (T4) are plotted. On Figures 2.3(c) and 2.3(d), the original 3D visualization is projected to a 2D view showing the exact destination ports more clearly, while hiding the destination IP address.



(a) 3D Original

(b) 3D Filtered

(c) 2D Original

(d) 2D Filtered

Figure 2.3: Destination IP and port from full source IP address.

Figure 2.3(a) shows a high number of source IPs probing a single port on the entire class C destination network and dense areas around low-order ports. Due to the large number of horizontal scans (probing a single port on all destination IP addresses as noted by bottom-left to top-right diagonal lines) displayed, a security analyst might have trouble identifying which scans warrant further analysis from Figure 2.3(a). However, most (if not all) horizontal scans in Figure 2.3(b) likely reveal some type of malicious activity. For example, some horizontal scans that only target the first 64 addresses of the dataset subnet, suggest that the scanner is aware of the network topology in question.

The data that was automatically removed in moving from Figure 2.3(a) to 2.3(b) was, as previously noted, classified as non-threatening by the EM filtering. For example, the left-most horizontal scan on Figure 2.3(a) belongs to a single source probing

all hosts on the destination network for port 32000 TCP which is not offered. On Figure 2.3(c), we notice a single source IP address attempting connections to a large number of TCP destination ports below 40000 (vertical line on the right). However, this traffic is unimportant because the source is probing a large number of destination ports on a non-existing host (note this traffic is absent from Figure 2.3(d)). Therefore, filtering network traffic using EM helps in reducing the volume of traffic to be investigated for possible malicious behavior.

Chapter 3

Analytical and Empirical Comparison of Two Scan Detection Algorithms: TRW and EM

In this chapter, we analyze and compare two network scanning detection techniques introduced in Chapter 2, based on remote host's successful or failed connection attempts: Threshold Random Walk (TRW) [45] and Exposure Maps (EM) [123]. While both techniques can be used in a broader context, one of their main applications is network scanning detection. We present the results of empirical experiments performed in both controlled and uncontrolled environments, using several metrics for comparison. In particular, we study the impact of the critical parameters for each technique in order to understand how to set these parameters according to network type, structure, and traffic. Our analysis shows the expected number of connection attempts required by the EM technique is substantially lower than the TRW technique but the former technique yields a higher rate of false positives. We also present what is, to our knowledge, the first automated mechanism to estimate a reference baseline of scanners (in a given dataset) for scan detection accuracy evaluation.

3.1 Introduction

Although numerous network scanning detection approaches have been proposed in literature, very few proposals offer both reasonably accurate and efficient detection. The high false positive and false negative rates inherent with these techniques have contributed to few being adopted by IDSs and they are rarely used to automatically block identified scanners. The majority of proposed scanning detection techniques depend on detecting abnormal network traffic in remote host traffic directed to the local network. Unfortunately, most detection approaches can be evaded easily by adversaries. The only exception appears to be features based on a remote host's successful or failed connection attempts (e.g.,TRW [45]) since the objective of network

scanning is to find open ports and thus it is assumed that the adversary does not know the open ports in the monitored network.

In this chapter, we study two scanning detection techniques based on a remote host's successful or failed connection attempts: (i) TRW [45], a fast online scan detection algorithm [45] that classifies remote hosts as either benign, scanner, or pending according to the ratio of remote host's successful or failed connection attempts in the inbound network traffic within a time window; and (ii) EM [123, 120], a technique that leverages the knowledge of active services in the monitored network to detect scanners from their first failed connection attempt.

We first present an analytical analysis of both algorithms, comparing them using the following metrics:

1. algorithm objective and scope;
2. required parameterization and configurability;
3. detection accuracy;
4. expected number of connection attempts before detection;
5. false positive rate;
6. computational resources;
7. resistance to evasion and attacks; and
8. scalability.

An empirical evaluation of both algorithms is then presented (also, see Whyte [120] for a basic evaluation of EM and TRW using the synthetic LBNL/ICSI dataset [83] and another non-synthetic dataset). We focus on detection accuracy in terms of detection rate, rate of false alarms, and efficiency. The evaluation uses real-world network traffic captured from two different network environments. In order to establish an estimated ground truth of scanners, we present a scan detection classification methodology that generates an estimated ground truth of scanners upon analyzing the full network traffic of remote hosts. Rather than relying on a binary classification

of remotes into non-scanner and scanner categories, in our methodology, remote hosts are classified into six categories ranging from benign to scanner.

In summary, our contributions include:

1. Analytical Evaluation We conduct an in-depth study of TRW, a state-of-the-art algorithm for scan detection that is often considered as close to a "gold standard" as there is in this area. We compare it with EM, another scan detection technique that has recently received attention as Outstanding Paper at ACSAC 2007. We show the effect of each algorithm parameter in terms of detection accuracy, detection speed (number of connection attempts observed before detection), and computational resources. We also study the scalability of both algorithms and their resistance to evasion and attacks.

2. Scan Detection Classification Methodology: We present a novel classification 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 two studied algorithms). The new methodology provides a reference baseline for evaluation for each dataset studied, in the absence of ground truth (we study this in-depth in Chapter 6).

3. Empirical Evaluation: We evaluate the detection accuracy of both algorithms by conducting several experiments with a variety of configurations on two datasets gathered from two different sites. We also show that the results of our experiments confirm the analytical comparison of the two algorithms.

Organization. Section 3.2 provides a detailed analytical comparison of TRW and EM discussing features, capabilities, and limitations of both algorithms. Section 3.3.1 describes the datasets used and their network environment. Section 3.3.2 presents a new methodology to obtain a reference baseline for evaluating network scanning detectors. Section 3.3.3 provides the results of the empirical evaluation of the TRW and EM algorithms in terms of the detection accuracy. Section 3.4 concludes.

3.2 Comparison and Analysis

In this section, we provide a detailed analytical comparison of TRW and EM discussing features, capabilities, and limitations. Our analysis considers a number of comparison metrics that provide a basis to quantitatively compare the two scanning detection techniques.

3.2.1 Objective and Scope

TRW scanning detection is based on examining the ratio of failed and successful connection attempts made by a remote host. Therefore, TRW is able to identify hosts exhibiting either benign or scanning behaviour. On the other hand, EM scanning detection is based solely on the observation of the number of failed connection attempts and thus it only identifies scanning behaviour. Specifically, a remote host initiating one or more failed connection attempts is classified as a scanner even if it makes successful connections before, within, or after the failed connection attempt(s).

Basic TRW [45] considers only unique destination hosts (i.e., local hosts in the monitored network). Thus, after the first connection attempt, all connection attempts (whether successful or not) to the same destination host (even to different ports) are ignored. This means that subsequent connection attempts from the same remote host to the same destination host will not change the TRW likelihood ratio of the remote host. Therefore, unlike EM, a remote can scan multiple ports on a single host (i.e., a vertical scan) without being classified as a scanner by TRW. The algorithm can be modified to consider failed connection attempts to individual ports (either TCP or UDP) or a destination host (e.g., see [118] and [72]) so that failed connection attempts to the same local host but to a different port/protocol will update the remote host likelihood ratio correspondingly. Such modifications, however, require more state to be maintained as the algorithm requires that both the probed port/protocol for each local host and the remote hosts that performed the probing be tracked. On the other hand, EM considers every connection attempt to IP/port/protocol not in the NEM table (even to the same port on the same host) without the need to keep track of the remote hosts that performed the scanning. Further discussion about resources requirements for each algorithm is discussed in section 3.2.6.

3.2.2 Parameterization and Configurability

TRW. Prior to deployment, TRW requires four parameters to be set: θ_0, θ_1, α, and β. These can also be set during operation to improve the performance of the scan detection. We now study how to choose these parameters by examining the effect each one has on the detection results.

1) **The probability a benign remote initiates a successful connection (θ_0):** because a benign remote host is supposed to know the destination host's domain name (or IP address) and the offered service prior to initiating a connection attempt, the probability of a successful connection from a benign remote host is expected to be high. However, there is no obvious criteria for choosing the right value for this parameter. The overall network architecture and the availability of services in some regards dictate the probability that a legitimate user may make a failed connection attempt. Thus, based on the characteristics of the network, the lower the probability that a legitimate user will make a failed connection attempt, the closer θ_0 should be set to 1. On the other hand, there are several cases where legitimate users could make failed connections. For example, a temporarily unavailable server, reallocation of a service, server IP reallocation without an immediate DNS update, and peer-to-peer misconfiguration, are all possible cases that require a lower θ_0 value. In analyzing some network traces, Jung et al. [43] found that for a given remote host the percentage of the local hosts for which the connection attempt with the remote host failed is either 0% or 100% (or very close to these numbers). They then suggested that hosts with a percentage of less than 80% be declared benign while hosts with a higher percentage are possibly scanners. Their assumption of considering remote hosts with a percentage greater than or equal to 80% as possible scanners (i.e., the remote host made failed connection attempts to at least 80% of the total local hosts this remote host attempted to connect with) was supported by independent output from a Bro IDS [2] on their datasets. Bro classified all IP addresses identified by TRW as either scanners[1] or

[1]The scan detection policy used in Bro keeps track of the number of local hosts a remote initiates connection attempts to according to the destination port. For some ports, it counts the number of distinct destination addresses and for the rest it counts only those where inbound connection attempts have failed. A remote host is flagged as scanner once this number exceeds a configurable threshold (set to 20).

sending Code Red/Nimda HTTP worm payloads (Bro uses signatures for known worms). The value chosen for the TRW parameter θ_0 affects the likelihood ratio Λ for all remote hosts. Recalling Equations 2.3 and 2.4 from Section 2.6.2, note that a larger θ_0 value increases the effect that successful connections contribute towards the likelihood ratio in classifying r as benign since Λ will reach the lower threshold η_0 faster. Likewise, if we assume that $\theta_0 > \theta_1$, a larger θ_0 value increases the effect that failed connections attempts have towards classifying that r is a scanner because Λ will reach the upper threshold η_1 faster. We further discuss how the value of θ_0 affects the number of required observations to reach a decision in section 3.2.4.

2) **The probability that a scanner initiates a successful connection (θ_1):** if we assume the adversary has no prior knowledge of the targeted network, the probability that a scanner initiates a successful connection will depend on the density of the offered services in the monitored network. This is based on the assumption that an adversary will probe destination IP addresses either sequentially or randomly in a specific subnet for the targeted services in order to find open ports. In general, this probability is expected to be very small since in most of today's networks the number of offered services is usually very small with respect to the total available IP addresses in a subnet and the possible services at each address. Considering only one protocol (e.g., TCP), θ_1 can be defined for a particular subnet as follows:

$$\theta_1 = \frac{\text{number of offered services}}{2^{16} * \text{ (number of IP addresses in the network subnet)}} \tag{3.1}$$

For example, in a very rich class C network (i.e., /24 subnet) where each local host has 10 open TCP ports, the chance of randomly finding a service (i.e., θ_1) is $\frac{254 \times 10}{2^{16} \times 254} = 0.00015$. However, the probability of having a port open is not the same for all ports. Specifically, there are a few common services that are offered in most networks. For instance, it is more likely to find port 80 (HTTP) open than other TCP ports (e.g., port 4). This implies, at least for some networks, θ_1 is in fact

greater than what was defined in equation 3.1. In order to choose an appropriate value for θ_1, a heuristic that takes both service popularity and network density into account is required. Recall from Section 2.6.2 that θ_1 has the opposite effect of θ_0. A larger θ_1 value decreases the effect of successful connections towards selecting H_0 as Λ will take longer to reach the lower threshold η_0 (assuming $\theta_0 > \theta_1$) and it decreases the effect of failed connections attempts towards selecting H_1 since Λ will take longer to reach the upper threshold η_1. The observation that a connection attempt is more likely to be a success from a benign source than from a scanner means that it is more likely that θ_0 is larger than θ_1.

3) **The desired detection probability (β):** both the upper threshold η_1 and the lower threshold η_0 depend directly on this parameter. The higher the value of β the higher the value of η_1 (see Equation 2.8 in Section 2.6.2) and thus the more failed connections made by a remote host r are required for its likelihood ratio Λ to reach η_1 (i.e., classified as a scanner). Similarly, the higher the value of β the lower the value of η_0 and therefore the more successful connections made by a remote host r are required for its likelihood ratio Λ to reach η_0 (i.e., classified as benign).

4) **The desired false positive probability (α):** similar to β, the false positive probability α affects both the upper threshold η_1 and the lower threshold η_0 but in the opposite way (see Equation 2.8 in Section 2.6.2). It is important to note, however, that a very low value of α will dramatically increase the value of η_1 which means TRW will require the observation of many failed connections to classify a source as a scanner regardless of β value. Similarly, choosing a very high value for β will dramatically decrease η_0 regardless of α.

One might argue that the ideal values are 1 for β and 0 for α respectively. However, the probability of detection varies with the false positive rate, and thus ideally TRW should be tunable to favour either the ability to detect scanners or to minimize false positives. The value of β is expected to be chosen (by the network administrator) very high since the algorithm's objective is to detect the actual scanners. The value of α, however, while always desired to be small, it should be set based on the false

positive rate experienced in the operational network. If for example the operational stance is to block detected scanners from accessing the network, a very low false positive probability should be set (by the network administrator) to reduce the chance of blocking potential legitimate users that are misclassified as scanners by TRW. Nevertheless, to our knowledge, there is no criteria in TRW to choose appropriate values for either α or β. Using the θ_0 and θ_1 parameters, the likelihood ratio Λ gets updated in the algorithm after each failed connection attempt initiated by the corresponding remote towards newly visited local addresses in the monitored network. Accordingly, when choosing values for α and β the values of θ_0 and θ_1 should be taken into consideration. Setting $\theta_0 < \theta_1$ implies the algorithm classification will work the opposite way because failed connection attempts decrease the likelihood ratio (since $\frac{1-\theta_1}{1-\theta_0} < 1$) and vice versa for successful connections. Having $\theta_0 = \theta_1$ means that the likelihood ratio will never get updated (since $\frac{1-\theta_1}{1-\theta_0} = \frac{\theta_1}{\theta_0} = 1$).

EM. There are no parameters to set in EM. As discussed in Section 2.7, a single connection to {IP address, port, protocol} not in the NEM is sufficient to classify a remote host as scanner. The construction of the vetted NEM, however, requires a training period to identify the offered services in a given network by passively observing a target network hosts' responses to incoming connection attempts (as mentioned in Chapter 2, an unvetted NEM does not require a training period during its construction). Moreover, once the NEM is constructed, compliancy with the network security policy is checked manually by network administrator. The appropriate duration for the training period is based on network structure and the nature of the traffic. While there is no obvious way to estimate this period, it is expected to last for the number of days the network administrator expects are needed for all or most of the offered services in the monitored network to be accessed. During ordinary network operation, passive observation of local hosts' responses is still required. If a new service is offered by a local host (i.e., a service not currently listed in the NEM), it will be discovered automatically and added manually to the NEM table only if the offered service complies with the network security policy (see Section 2.7.1). In contrast, TRW does not require a training period and it can be used directly to detect scanners. Nevertheless, setting each of the TRW parameteres to a practical value requires a priori knowledge

of the network in question. Typically, such knowledge might need to be acquired over time (i.e., a training period).

Summary. Although TRW is configurable through a set of detection accuracy parameters, there is no known criteria to choose appropriate values for these parameters. Our analysis shows that TRW detection accuracy is sensitive to its parameters and that it is hard to predict a suitable values for θ_0 and θ_1 according to the deployment environment. In contrast, EM has no parameters to set which limits the ability to tune the detection accuracy of the algorithm according to the monitored network environment. Also, unlike TRW, EM requires a training period, prior of being able to detect scanners, to identify the active services in the monitored network.

3.2.3 Scanning Detection Accuracy

TRW. The algorithm depends on the assumption that scanners have a higher percentage of failed connection attempts than benign hosts. If this assumption does not hold for a remote host, a higher chance of misclassifying the host as a scanner can be expected. As discussed in section 3.2.2, the chosen values for the TRW parameters can have a great effect on its accuracy. In addition, one may assume that both the network architecture and the amount and type of incoming traffic will change over time. The dynamic nature of networks requires periodic updating of the parameters to maintain an acceptable detection accuracy rate. Once a host is classified as either a scanner or benign, there is no further check on the traffic associated with that host as classification is complete. In practice, it is likely that the state of a host could change from benign to malicious (e.g., due to a new worm infection) and vice versa (e.g., because a system patch is installed). While the TRW algorithm can be easily changed by introducing a time window such that the remote host state is reset to the *pending* state after a predefined time period since the last classification, choosing an appropriate time window once again depends on the network structure and traffic. Although TRW is designed to be used for only TCP based network traffic, the algorithm can be extended to consider UDP and ICMP traffic (see [118, 72]). One drawback of using TRW for UDP traffic is that the time required to know if the connection is unsuccessful could be long enough for an adversary to scan further IP

addresses of the network in question before being caught (i.e., before TRW classified it as a scanner).

EM. Every connection attempt to an {IP address, port, protocol} not in the NEM table is flagged as an atomic scan event. Thus, a remote host could be classified as scanner from the first connection attempt to an {IP address, port, protocol} not in the NEM table. EM detection does not rely on tracking this activity of remote hosts. This minimizes the amount of state that has to be maintained by the EM system, eliminating the need to maintain any type of sliding time window as part of the detection algorithm. Accordingly, the ability to detect and record atomic scan events enable the detection of very slow scanners. For example, adversaries might perform targeted or distributed scans (e.g., as a part of a botnet) in a slow or stealthy way to evade detection. A penalty of such strict detection, however, is a possible high false positive rate as we discuss in section 3.2.5. EM requires at least one connection attempt to identify a remote as a scanner. The only way for a remote to evade detection by EM is for a remote host to somehow only probe offered services in the network.

Summary. In theory, EM detects all scanners except those that make connection attempts to active services in the monitored network. However, as we discuss in Section 3.2.5, this also means that EM has a high false positive rate. In contrast, using TRW default parameters, TRW has a more conservative detection strategy in which at least four consequent failed connection attempts initiated to four distinct local hosts are required (within a given time window) for a remote to be classified as a scanner. Therefore, EM is expected to have a higher detection accuracy and to detect more stealthy scanners than TRW. However, note that it is possible to configure TRW parameters such that it detects scanners from their first unsuccessful connection attempt, though this would generate high false positive rate (higher than that of EM) as we discuss further in Section 3.2.5.

3.2.4 Expected Number of Connection Attempts Before Detection

TRW. In case the adversary possesses no priori knowledge of the monitored network, the approximate expected number of connection attempts N (whether successful or

not) before a scanner is detected is (see [45] for a detailed explanation of how to obtain the conditional expectation):

$$E[N|H_1] = \frac{\beta\ ln\frac{\beta}{\alpha} + (1-\beta)\ ln\frac{1-\beta}{1-\alpha}}{\theta_1\ ln\frac{\theta_1}{\theta_0} + (1-\theta_1)\ ln\frac{1-\theta_1}{1-\theta_0}} \tag{3.2}$$

In a similar way the approximate expected number of connection attempts N (whether successful or not) before a host is classified by TRW as benign is:

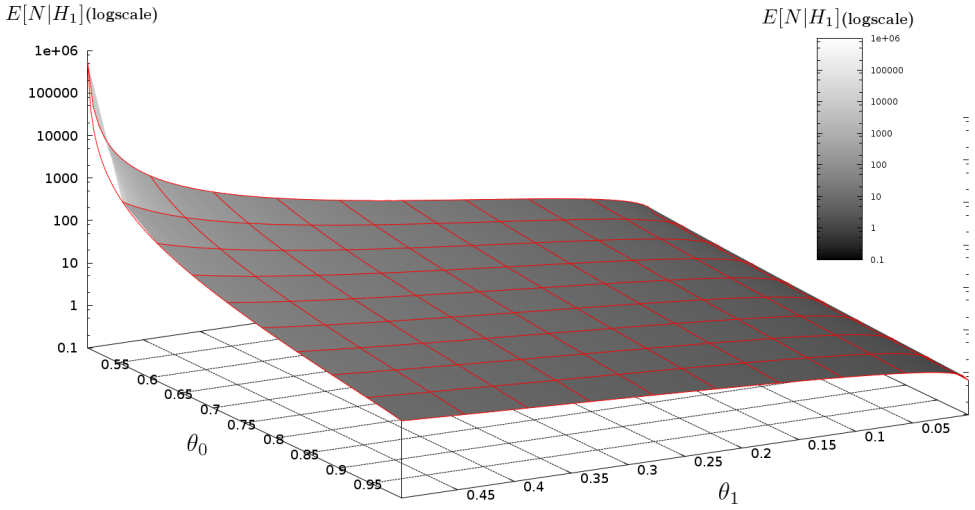
$$E[N|H_0] = \frac{\alpha\ ln\frac{\beta}{\alpha} + (1-\alpha)\ ln\frac{1-\beta}{1-\alpha}}{\theta_0\ ln\frac{\theta_1}{\theta_0} + (1-\theta_0)\ ln\frac{1-\theta_1}{1-\theta_0}} \tag{3.3}$$

The values selected for all four TRW parameters (θ_0, θ_1, α, β) affect the expected values above. Figure 3.1(a) shows how $E[N|H_1]$ changes as θ_0 and θ_1 change. $E[N|H_1]$ increases with decreasing θ_0 and with increasing θ_1. For example, for $\theta_0 = 0.6$ and $\theta_1 = 0.4$ ($\alpha = 0.01$ and $\beta = 0.99$), $E[N|H_1] = 56$, whereas for $\theta_0 = 0.52$ and $\theta_1 = 0.48$, $E[N|H_1] = 1,407$. Figure 3.1(b) shows how $E[N|H_1]$ changes as α and β change respectively. As we discussed in Section 3.2.2, the higher the value of β the higher the value of η_1 and thus the higher the value of $E[N|H_1]$ (i.e., more failed connections made by a remote are required before being classified as a scanner). Also, the lower the value of α the higher the value of η_1 and the higher the value of $E[N|H_1]$. For example, for $\alpha = 0.001$ and $\beta = 0.95$ ($\theta_0 = 0.8$ and $\theta_1 = 0.2$), $E[N|H_1] = 8$, whereas for $\alpha = 0.3$ and $\beta = 0.7$, $E[N|H_1] = 1$.

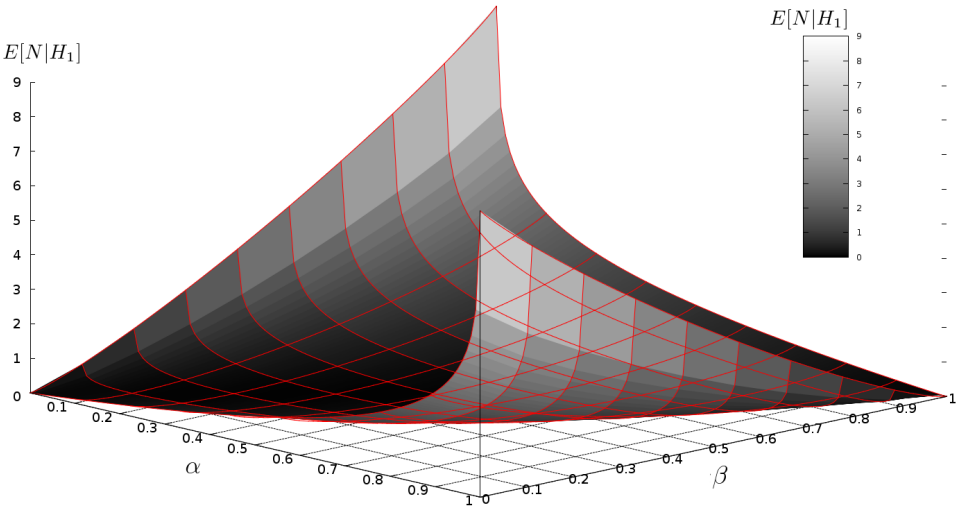
EM. The EM algorithm is faster in detecting scanners (in terms of the number of connection attempts that a scanner can perform before being detected) since successful connections are not considered as part of the detection algorithm. Assuming that an adversary targets only one protocol (e.g., TCP), the approximate expected number of connection attempts N (whether successful or not) before a scanner is detected by EM is:

$$E[N|H_1] = \left(1 - \frac{\text{number of NEM entries}}{2^{16} * \text{ (number of IP addresses in the network subnet)}}\right)^{-1} \tag{3.4}$$

This value is expected to be approximately 1 in most cases. For instance, as in our



(a) For different values of θ_0 and θ_1 ($\alpha = 0.01$ and $\beta = 0.99$)



(b) For different values of α and β ($\theta_0 = 0.8$ and $\theta_1 = 0.2$)

Figure 3.1: The expected number of connection attempts (n)

previous example, in a very rich class C network where each local host has 10 open TCP ports, $E[N|H_1] = \left(1 - \frac{254 \times 10}{2^{16} \times 254}\right)^{-1} \approx 1$. Thus using EM for scan detection, a scanner is expected to be detected from its first connection attempt to the monitored network.

Summary. The expected number of a remote's connection attempts before TRW classifies the remote is determined by TRW four parameters θ_0, θ_1, α, and β (with the default values, $E[N|H_1] = 5.4$). On the other hand, with EM, the expected number depends on the number of IP addresses and the number of active services in the monitored network (in the majority of network environments, $E[N|H_1] \approx 1$). Note that, unlike TRW, successful connection attempts are not considered in EM.

3.2.5 False Positive Rate

There are several cases where benign hosts could initiate failed connections. For instance, a misconfiguration in the targeted network (e.g., a reallocated server to a different IP address without updating the corresponding DNS entry) or misconfiguration in the remote host initiating the connection (e.g., an application in the remote host pointing to a wrong destination IP address). There are also cases of applications that need to legitimately perform scanning in the monitored network either from within the network (e.g., searching for a network printer by simply scanning the network on a specific port) or from external sources (e.g., web crawler). In both algorithms, failed connection attempts made by benign remote hosts could lead to false positives.

TRW. False positives occur when a benign host is misclassified as scanner (i.e., hypothesis H_1 is chosen when r is a benign host). Although the tolerated false positive probability (α) can be adjusted, TRW might generate many false positives if the other parameters (i.e., θ_0, θ_1, β) are not chosen carefully. To be misclassified as scanner, a misconfigured benign host needs to make n failed connection attempts (assuming the worst case with no successful connections in between) to n unique destination IP addresses in the monitored network:

$$\left(\frac{1-\theta_1}{1-\theta_0}\right)^n \geq \frac{\beta}{\alpha} \qquad \text{(see Equations 2.3 and 2.4)}$$

$$n \ \log\left(\frac{1-\theta_1}{1-\theta_0}\right) \geq \log\left(\frac{\beta}{\alpha}\right) \qquad \text{(assuming that } \beta \geq \alpha\text{)}$$

$$n \ \geq \left\lceil \frac{\log\left(\frac{\beta}{\alpha}\right)}{\log\left(\frac{1-\theta_1}{1-\theta_0}\right)} \right\rceil \tag{3.5}$$

Figure 3.2 shows how n changes for different values of α and β. As discussed in section 3.2.2, n increases as β and θ_1 increase and as α and θ_0 decrease.

EM. A remote host initiating a single failed connection attempt will be classified as scanner by (basic) EM even if there are successful connections from the same remote host before or after the failed connection attempt. Therefore, EM is potentially susceptible for a high false positive rate in comparison with TRW. The higher the probability benign hosts make failed connections, the higher the false positive rate. For example, if a legitimate remote user using SSH client for logging types a wrong IP address, EM will classify the user's remote host as scanner. Using a vetted NEM in EM, a connection attempt to an IP/port/protocol not in the NEM is considered an atomic scan event even if the connection is successful. While this feature has an advantage of generating alarms for ports that are not supposed to be open and thus connections from remote hosts to these ports will be counted as undesirable traffic, false positives will be generated whenever a new legitimate service is added to the network after the training period or when an existing service is used which was not accessed during the training period. Using an unvetted NEM[2], however, a successful connection attempt to an IP/port/protocol not in the NEM will be added to the NEM and thus the remote host initiating the connection will not be classified as scanner for future connections to the same NEM entry. One advantage of EM (either using a vetted or unvetted NEM) is that it will not generate false positives in case some of the offered services in the monitored network are temporarily inactive since connection

[2]The *unvetted* NEM table will always be updated whenever a new service is offered in the monitored network. This is useful in case of an open security policy in which all or most possible network services can be offered at anytime and thus a training period is not required (see Section 2.7.1).

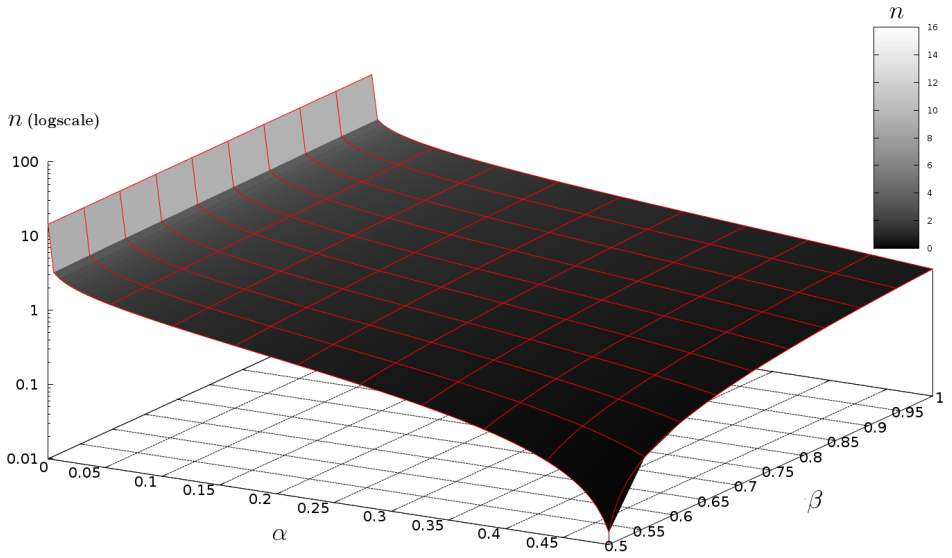


Figure 3.2: The minimum number of successive failed connection attempts n (with with no successful connections in between) for different α and β before TRW classifies a host as scanner ($\theta_0 = 0.8$ and $\theta_1 = 0.2$)

attempts will be checked against the NEM and not whether they are successful or unsuccessful. In contrast, in TRW, a temporary outage at a destination network at which some contacted hosts become unresponsive could lead to false positives.

Summary. While TRW false positive rate is adjustable in a trade-off with the detection rate, EM lacks this flexibility. Also, considering Equation 3.5, TRW has potentially a lower false positive rate relative to EM (e.g., with TRW default parameters, $n = 4$ while $n = 1$ in EM). However, unlike TRW, failed connection attempts to previously offered network services are not considered in EM which helps to reduce the number of false positives.

3.2.6 Required Computational Resources

TRW. In this section, we analyze both the required memory/disk space and the required computation time (for updating the algorithm data structures and scores for each connection attempt). TRW maintains 3 variables for each remote host contacting

the target network: the remote host state (which is one of *pending*, H_0, or H_1), the likelihood ratio (Λ) for that host, and the set of distinct local hosts previously contacted by the remote host. Let L be the number of available local IP addresses and R be the number of remote hosts contacting the monitored network. Assuming that 4 bytes are needed to store one IPv4 address, 1 byte to store remote host state, and 1 byte to store remote host likelihood ratio value, the maximum required space for TRW is $R(4L + 1 + 1)$ bytes. For example, for a class B network (i.e., $(2^{16} - 2)$ local IP addresses) and assuming all public Internet addresses contacted the network (i.e., the worst case: 2^{32} IPv4 address space minus 2^{28} private networks and multicast addresses), TRW requires at most $(2^{32} - 2^{28})(4(2^{16} - 2) + 2) = 960$ terabytes. IPv6 has a much larger address space ($\sim 2^{128}$) than IPv4 and thus the maximum required space by TRW in an IPv6 network is vastly larger.

For each new flow, a lookup is required to find out if the {source IP address, destination IP address} tuple exists. TRW maintains two sets: a *failed connections* set indexed by the IP address pair of the scanner IP address and the unique host IP address it has scanned, and a *successful connections* set indexed by the IP address pair of a remote host IP address and the unique host IP address it has made successful connections with. If the current flow's IP address pair does not exist in both sets, it will be added to one of these two sets according to whether the connection is successful or unsuccessful. The processing time for such a lookup (and insertion operation if required) depends on the data structure used to store these sets and the size of each set. In case we assign a lifetime to each tuple (in either set), tuples need to be deleted when their lifetime expires. This requires traversing each set periodically (this is a possible extension to TRW so that after a specific time window the remote host state is set back to pending for reevaluation).

It is also important to note that TRW needs to 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 (possibly a few minutes according to the operating system type

and version). 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 in the local host is open and thus the connection is successful. Otherwise, if there is no UDP reply from the local host for a specific time (possibly a few minutes according to the operating system type and version) the connection is considered unsuccessful.

EM. The NEM table is the only data structure that (basic) EM needs to maintain in order to perform scanning detection. The size of this table is simply the number of services offered on the network, S. If EM is used to perform some form of active response option as a result of scanning detection, all scanner IP addresses will need to be stored (possibly for a predefined period of time). Using our previous example, if there are 1000 services offered in the network and assuming all public IP addresses made failed connection attempts to the network, the required storage space to perform scanning detection is simply $3*1000) = 3$ kilobytes (assuming 2 bytes representing the port number and 1 byte representing the protocol in the NEM). If an active response option is used, the required storage space is $4*(2^{32}-2^{28}) + 3*1000 = 3.75$ gigabytes (the first term is for the scanner list and the second term is for the NEM table). For each new flow only one lookup is required by EM to determine if the remote host's {IP address, port, protocol} tuple matches an entry in the NEM. In practice, the size of the NEM is expected to be a relatively small and could fit in main memory to increase the speed of lookup operations. A failed connection implies adding the remote host IP address to the scanner list.

In the case of an unvetted NEM, the IP address of a remote host making a successful connection to a service not in the NEM will be added to the scanner list. In order to be able to delete its IP address from the scanner list once the connection is determined successful, a temporary data structure is required to hold remote hosts' IP addresses of undetermined active connections (i.e., whether successful or not) for IP addresses that do not already exist in the scanner list. Consequently, an additional overhead of a lookup operation in the scanner list is required for any connection attempt with no corresponding NEM entry. However, such operations will be required in order to initiate active response options to the network traffic of scanners.

Summary. Both the required memory space and the required computation time is significantly lower in EM relative to TRW.

3.2.7 Resistance to Evasion and Attacks

In this section, we discuss the algorithms' resistance to some possible evasion and gaming tactics and to the denial of service (DoS) attack.

Evasion

TRW. The TRW algorithm credits remote hosts for making successful connections by moving their corresponding likelihood ratio towards the *benign* state. This property can be exploited by adversaries in two ways. First, it can be used to increase the number of allowed failed connection attempts before TRW classifies the remote host as scanner. Specifically, an adversary C, with knowledge of some available services in the target network (e.g., knowing the URL of a web server in the target network), can make successful connections to these services while scanning the network to postpone or evade detection. However, basic TRW considers only the first successful connection to a local IP address hosting a service (i.e., subsequent successful connection events to the same destination IP address will not change the likelihood ratio Λ assigned to the remote host). Secondly, TRW classifies a remote host as benign once the remote host's likelihood ratio goes below a certain threshold. C can use the fact that successful connection attempts decrease the likelihood ratio and vice versa by first making connections to only available services that C has knowledge of in the monitored network. Using this strategy, the likelihood ratio reaches the benign threshold and thus subsequent failed connection events, which represent the actual scanning, will not be checked by TRW (at least for a predefined time window).

Kang et al.gm [46] introduced a distributed scan attack that takes advantage of the above TRW limitation. Assuming C has access to a set of IP addresses, he starts off the scanning campaign with one of these IP addresses. Eventually, TRW will classify this IP address as scanner after a specific number of failed connection attempts that C might guess (a conservative guess is biased towards a small number). Considering that this IP address might then be blocked by the target network, C continues scanning

using a different IP address. This time the knowledge of the discovered services in the target network by the previous IP address can be employed to further delay being declared a scanner by TRW. This process may continue until the complete address space of the target network is scanned or when C runs out of fresh IP addresses that can be used for scanning. However, in the latter case C could wait until some of the IP addresses are unblocked by the target network and then resume scanning, given C's time constraint will allow such a delay. Using this attack, the authors [46] showed that the upper bound for the required number of IP addresses that need to complete scanning the address space of a monitored network is a logarithmic function proportionate to the size of the address space being scanned. They also proposed a hybrid approach combining both TRW and a timeout-based scan detection method which simply counts the number of unsuccessful connections attempted in a fixed time window in order to check if a threshold is exceeded (similar to the approach of Kato et al. [48]). Although this hybrid approach can be easily evaded by making connections with enough time delay between them to avoid reaching the threshold, this approach imposes a time constraint on C. In fact, the authors showed that using a delayed distributed scan against this approach, the number of source IP addresses C requires will be proportional to the address space size. Nevertheless, this hybrid approach is susceptible to a higher false positive rate [46].

As mentioned in Section 3.2.1, since TRW considers only the first failed connection between a pair of IP addresses (ignoring all subsequent unsuccessful connection attempts from the same remote host to the same destination host), scanning all ports on one local host by one remote host will update the remote's likelihood ratio only once. Therefore, C can optimize the scanning process such that for any particular host in the target network only one remote host is used to scan all its ports for different services.

As discussed in Section 3.2.6, in order to determine that the connection failed to establish, TRW needs to wait for the protocol to timeout (e.g., 2 minutes is the default timeout value for TCP). This gives C using an automated scanning tool (e.g., NMAP [79]) plenty of time to perform scanning against the monitored network before being detected by TRW, as noticed by Schechter et al. [95]. Under normal conditions,

open network ports will reply immediately to C's scanning requests which enables C to gather information about a targeted network's available services before being detected. In fact, if the connection rate is not monitored by other means (e.g., an IDS), C's scanning process is bounded only by the target network and the network bandwidth between C and the target network.

EM. Remote hosts obtain no reward for making successful connections in the EM scan detection technique. Thus, even if C has a priori knowledge of some of the open ports in the target network, the first failed connection attempt means the remote host will be labeled as a scanner by EM. In scan detection the threat model assumes that an attacker's motive is to gain information about active hosts in the target network and the services they are running. Knowing this information in advance precludes the need to scan the network in the first place. By this reasoning, EM is immune to evasion attack. However, in practice we might see an attacker with access to many IP addresses scan the target network using only a subset of these addresses. Any follow-on attacks can be launched from new remote IP addresses against the previously found open ports.

Denial of Service

TRW. It is feasible for an adversary C with limited resources (even one remote host) to perform a complete DoS attack against a target network if a blocking policy is in effect for remote hosts that are labeled as scanner. The attack is simply sending IP packets with spoofed source IP addresses to a non-existent host or to a host on a closed port in the target network. TRW classifies a remote host as a scanner after the observation of failed connections to n unique local hosts. Therefore, to block an address space of s, C needs to send sn packets. If C has knowledge of some of the available services in the target network, C can send IP packets with the spoofed source IP addresses of a subset (or all) of the global IP address space to these available services. TRW will then classify the IP addresses C used to initiate these bogus connections as benign and no further scanning events will be detected.

Another possible DoS attack involves resource exhaustion. As we discussed in Section 3.2.6, TRW needs to track the different local IP addresses each remote host

attempted to contact. C could use the fact that TRW needs to maintain a large amount of state by launching the previous attack with a large number of spoofed source IP addresses. This would force the allocation of a large amount of storage space to keep track of meaningless data. If enough spoofed scan attempts were sent, all storage space may be exhausted causing system instability. C could then start the actual scanning campaign in the hopes that TRW will fail to detect the real scanners. Even if the storage space was sufficient to survive the attack, the computational capability (see Section 3.2.6) of the system hosting TRW could be exhausted or adversely affected. In this way, C could prevent TRW from detecting the actual scanning of the target network.

EM. The DoS attack is easier with EM scan detection. C needs only to send s packets to block an address space of size s since EM classifies a remote host as scanner from the first failed connection. However, EM is much less susceptible to resource exhaustion attacks (either disk or CPU resources). As discussed in Section 3.2.6, EM does not need to keep a state for remote hosts (other than scanners IP addresses) and the processing time for each new connection requires only a single NEM lookup.

Summary. For both TCP and UDP, the time required to determine whether a connection is successful or unsuccessful is long enough for a scanner to scan many IP addresses. Since EM relies on the NEM table to immediately determine the state of connections, only TRW is vulnerable to this attack. Also, only TRW is vulnerable to the attack of delaying detection by scanners that intentionally make successful connections. Both algorithms are vulnerable to distributed scan campaigns and to DoS attacks using spoofed remote IP addresses.

3.2.8 Scalability

TRW. As we have illustrated in section 3.2.2, setting the TRW parameters requires having a priori knowledge of the monitored network. Some network environments might be large or distributed which makes it difficult to acquire such knowledge from simply analyzing the network architecture and the volume of network traffic. Furthermore, for dynamic, transient, or diverse networks (e.g., where new hosts may be

added either for the short or long term, host mobility is possible, and there are less restrictions on the client applications or services run by hosts), a periodical reparameterization of TRW variables might be required to enable faster detection and a lower false positive rate.

As the size of the monitored network increases, the amount of state TRW requires increases. In fact, for large networks, TRW may require very considerable state to keep track of remote hosts' IP addresses together with the corresponding contacted local IP addresses (as discussed in section 3.2.6). The lookup and insertion operations within the data structures maintained in state might also be computationally expensive. The storage and computational capabilities required for the machine hosting TRW will be network dependant as the amount of state required to track connection activity will depend on the number of internal hosts, as well as the volume of external connection activity from remote hosts. Additionally, if active response actions are instrumented as part of the detection algorithm, this may also have an impact on the required system resources as well as performance. For example, if we consider IPv6, the required TRW state could increase by a factor of 2^{96}.

EM. For a vetted NEM, EM requires a sufficient training period to gather information about open ports in the monitored network. For large, distributed, dynamic, or transient network environments, this process would not only take considerable time but also requires frequent updates of the NEM entries that might not be captured passively (i.e., rather than using a training period to update the NEM, the IP space of the local network is scanned automatically on a consistent basis to capture changes in network services). Additionally, it might also be hard to check the NEM compliance with the network security policy (e.g., if the policy is not well documented). In such environments, an unvetted NEM might be more appropriate as the NEM table is built on the fly, adding any identified new service as it is discovered. While an unvetted NEM does not require a training period to populate it, it is not clear when to drop entries from the NEM table over time to account for services no longer offered. One suggested solution is to keep track of the last access time of each NEM entry when processing incoming connection attempts in order to drop entries that have not been accessed within a predefined period [120]. However, choosing the right period relies on

a number of factors, making selection of the appropriate value difficult. For example, the service type, location, as well as volume, size, and structure of the network might need to be considered.

Although the size of the NEM table is expected to increase with the size of the network, especially in dynamic environments where unvetted NEM is used, the NEM table size is relatively small and easily fits into main memory. The time required to do a NEM lookup (for each network flow) is an O(1) operation even in large networks. EM also adds the remote host IP address to the scanners list if the connection attempt does not match an entry in the NEM table. As discussed in section 3.2.6, the maximum size of the scanners list is the size of the IPv4 address space, at most $4 * (2^{32})$ bytes ($4 * 2^{128}$ bytes for IPv6).

Summary. The larger the size and the more the diversity of the monitored network, the harder it is to parameterize TRW and the larger the required memory space to keep its state. In contrast, with EM, the larger the size of the monitored network, the longer the required training period and the harder it is to maintain the NEM table.

3.3 Empirical Evaluation

In this section, we evaluate the performance of the TRW and EM scan detection techniques in terms of detection accuracy. This section first gives an overview of the used datasets and the associated network environment. It then describes our methodology in evaluating the performance of TRW and EM, and discusses the experimental results from a set of tests on TRW and EM.

3.3.1 Overview of Datasets

Dataset I. Dataset I is a full capture network trace collected at a class C university network with 62 Internet-addressable IP addresses. The network trace was gathered over the period from January 28 to March 13, 2007 (45 days). The size of the dataset is 41 gigabytes. While Section 3.3.2 uses this full capture, Section 3.3.3 uses the subset of this trace over the period of January 28 to March 10, 2007. Thirty active IP addresses during the capture period were observed. The network firewall

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

Table 3.1: Datasets statistics (dataset I of Jan 28 to Mar 13, 2007; dataset II of Jun 17 to Jul 4, 2010; (i) and (ii) are percentages of b; only remotes initiating TCP connections are counted in c).

allows inbound connection attempts to closed ports and unassigned IP addresses. The firewall responds to inbound connection attempts that are sent to IP address port combinations not in the access control list with RST packets. Note that 95% of inbound TCP connections in Table 3.1 are rejected (i.e., RST packet is sent by the destination) suggesting a high-volume of the overall network traffic is scanning activity. A few IP addresses in the network used P2P file sharing causing short bursts over the log capture period. Connection attempts to unavailable peers contributed to the observed 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 [27]. Six open ports (on 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 network trace was gathered over the period of June 14 to July 4, 2010 (21 days). The size of the dataset is 236 gigabytes. Note that Table 3.1 only shows the subset of this trace over the period of June 17 to July 4 which we use in the analysis in Section 3.3.2, while Section 3.3.3 uses the full capture. The number of observed active IP addresses during the capture period was 223. Inbound connection

attempts to closed ports or unassigned IP addresses were not allowed by the network firewall. Approximately 70% of inbound TCP connections are timed-out (did not go through the firewall) suggesting a high-volume of network scanning traffic.

Network protocols running on the open ports were identified by the same signature-based method used in the first dataset. Only protocol signatures that use the first few bytes of the TCP payload data with 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 (as the local machines in this dataset were for public use and P2P clients are not installed).

3.3.2 Evaluation Methodology: Identification of Scanners

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).

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 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 (see further discussion in Chapter 6). 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 (see further discussion in Chapter 6), which we call *ground truth reference* (*GTR*).

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 for this evaluation to establish a GTR of scanners. Although monitoring network traffic over a relatively long period of time (e.g., few days) provides 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 classification of 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 at 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

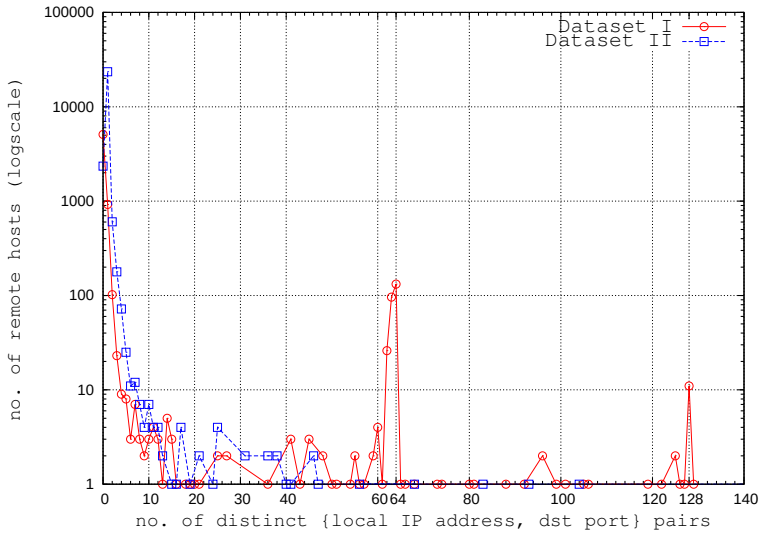


Figure 3.3: Number of remote hosts vs. the number of distinct {local_IP_addr, dst_port} tuples that they initiated failed connection attempts to, over two datasets. (y axis in log scale; best viewed in color)

site) contact the same network is low. Therefore, considering the change of state is less important (and not done) in the present classification.

For the two datasets in Section 3.3.1, we attempt to generate a GTR for each remote host based on the following metrics:

1. the number of distinct {local_IP_addr, dst_port} tuples that the remote host initiates successful connection attempts to over this same period;
2. the number of distinct {local_IP_addr, dst_port} tuples 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 3.3 shows the number of remote hosts for each number of distinct {local_IP_addr, dst_port} tuples that these remote hosts made unsuccessful connection attempts to. Approximately 78% (5,092 of 6,562) and 9% (2,351 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 varies widely between the two datasets (14% (922 of 6,562) and 88% (23,542 of 26,859) in the first and second datasets respectively). The difference between the two datasets is due to several possible reasons including: the number and type of offered network services and the volume and nature of scanning activities, especially considering the different dates these datasets are captured (for example, given that dataset II is more recent, it is more likely that it is under more distributed and stealthy scanning campaigns). Manual inspection of random sample of hundreds of these unsuccessful connection attempts reveal no correlation or similarity (e.g., different destination port numbers were targeted). Therefore, given that we have no access to these remotes' networks, we can only speculate about the possible causes of the high percentage in the second dataset. 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 3.4 plots the cumulative distribution for the number of remote hosts over the total number of distinct {local_IP_addr, dst_port} tuples that these remote hosts initiated failed connection attempts to.

Note that more than 91% and 96% of remote hosts made at most one failed connection 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_addr, dst_port} tuples 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.

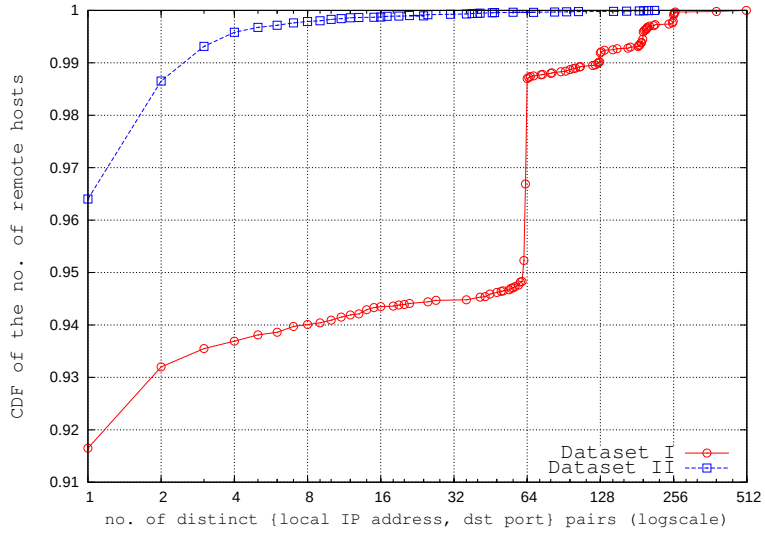


Figure 3.4: Cumulative distribution for the number of remote hosts and the total number of distinct {local_IP_addr, dst_port} tuples that they initiated failed connection attempts to, over two datasets. (x axis in log scale; best viewed in color)

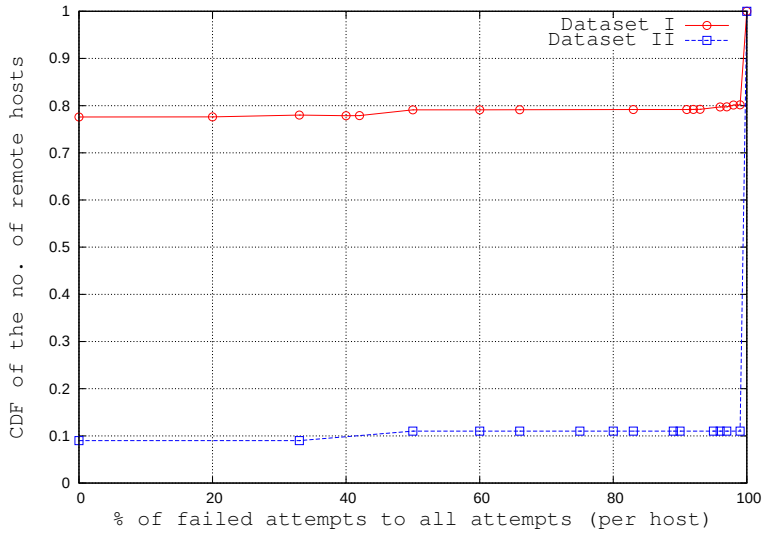


Figure 3.5: Cumulative distribution for the number of remote hosts and the ratio of the number of distinct {local_IP_addr, dst_port} tuples that each of these remotes initiated failed connection attempts to vs. the total number of distinct tuples the remote contacted either successfully or unsuccessfully.

$\boldsymbol{R_{inbound_S}}$	the number of distinct {IP_addr, dst_port} tuples the remote host R initiated successful connection attempts to.
$\boldsymbol{R_{inbound_F}}$	the number of distinct {IP_addr, dst_port} tuples the remote host R initiated failed connection attempts to.
$\boldsymbol{R_{outbound}}$	the number of distinct {IP_addr, dst_port} tuples that initiated connection attempts (whether successful or unsuccessful) to R.
$\boldsymbol{R_{\phi}}$	$R_{inbound_F}/(R_{inbound_S} + R_{inbound_F})$

Table 3.2: Notation for the classification criteria

Jung et al. [45] 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_addr, dst_port} tuples that the remote host initiated failed connection attempts to vs. the total number of distinct tuples the remote contacted either successfully or unsuccessfully.

Figure 3.5 plots the cumulative distribution of this ratio for all remote hosts except those 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 GTR 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_addr, dst_port} tuples that a remote host unsuccessfully contacted and the number of distinct {IP_addr, dst_port} tuples that the remote host successfully contacted. Table 3.3 gives the classification criteria for remote hosts (the notation is given in Table 3.2). This classification criteria consists of a set of heuristics extracted manually by analyzing several datasets, including those described in Section 3.3.1.

Class	Heuristic Rules
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

Table 3.3: Classification criteria for remote hosts

To classify a remote host R as *benign*, R must make successful connections with at least three distinct {IP_addr, dst_port} tuples and failed connection attempts with at most one tuple. The host will also be considered benign if both at a remote host Rleast one local host initiates an outbound connection attempt to R and R does not make failed connection attempts with more than one tuple. R is classified as *likely benign* if it makes only successful connection attempts with at least 75% of distinct {IP_addr, dst_port} tuples 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 tuples. Also, there should be no outbound connections (whether successful or unsuccessful) made to these remote hosts from any local host. This relatively strict heuristic is based on the assumption that it is unlikely for a benign remote to make only failed connection attempts with three or more services in the monitored network, given that there were no outbound connections to the remote. If R unsuccessfully contacts at least two distinct tuples 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 tuples or if there is at most one outbound connection attempt to R (note that if R is marked as a likely scanner and $R_{inbound_S} = 2$, then $R_{inbound_F}$

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 3.4: GTR Classification of remote hosts (dataset I of Jan 28 to Mar 13, 2007; dataset II of Jun 17 to Jul 4, 2010).

must be at least 6). While the *scanner* heuristic captures the typical scanning pattern (including stealthy scanners) of probing non-existing network services, 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_addr, dst_port} tuple and there is no outbound connections to them. Those remotes are hard to classify since failed attempts with one network service is not enough evidence to confirm 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 remotes 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 $R_{inbound_S}$ (see Table 3.2) relies 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 is the case in both datasets we study). The more local IP addresses offering the same network service (i.e., the same open port number), the higher the probability of a scanner of this port making successful connections. Therefore, the absolute number of successful connections to

unique local IP addresses may not accurately reflect a benign intent. To accurately weigh successful inbound connection attempts in this case, each destination port p is assigned a weight from 0 to 1 based on the p density in the target network as follows:

$$weight_p = 1 - \frac{\text{number of local hosts with port } p \text{ open}}{\text{number of local IP addresses}} \tag{3.6}$$

Each successful inbound connection attempt is now assigned the weight of the corresponding destination port. 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.

Therefore, we redefine $R_{inbound_S}$ as follows (n is the number of distinct {IP_addr, dst_port} tuples the remote host R initiated successful connection attempts to, and $weight_p^i$ is the destination port weight of the successful connection i according to Equation 3.6):

$$R_{inbound_S} = \sum_{i=1}^{n} weight_p^i \tag{3.7}$$

Table 3.4 shows the classification results of both datasets (note that $R_{inbound_S}$ is calculated according to Equation 3.7). The differences 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. For example, note that for dataset II, most of the remote hosts (about 80%) made only failed connection attempts with a unique {IP_addr, dst_port} tuple, causing the likely benign category to have only about 8%, relative to about 72% in dataset I.

Note that while using only two datasets could be a limitation since they are not guaranteed to be representative of all forms of network scanning, we believe that the datasets relatively long capture periods and the fact that these datasets are collected from quantitatively two different operational environments give an adequate generality for evaluating these particular algorithms. We emphasize, however, that using more datasets for evaluation provides better soundness and generality for the empirical evaluation.

3.3.3 Implementation and Results

We used the TRW implementation of the Bro language (TRW policy in Bro 1.4 NIDS [2], identical to Algorithm 1). We also implemented EM as a policy in Bro (identical to Algorithm 2). Both algorithms were configured to monitor remote hosts' behaviour over a one day time window (rather than the TRW default of 30 minutes). The write-expiry intervals for TRW scanners list S, TRW benign list B, and EM scanners list S were removed to keep track of all flagged remotes over the entire dataset capture period. TRW default parameters are used (see Algorithm 1). While choosing TRW default parameters is because these values have been shown to be reasonable in common network settings [43], it is important to note that another set of values might yield a better detection accuracy. However, there is no known criteria (to our knowledge) to choose appropriate values according to the operational environment. Using the ground truth reference of Section 3.3.2, we measure the performance of both algorithms using the following metrics:

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

$$\textbf{TP rate} = \frac{\text{number of true pos.}}{\text{number of true pos.} + \text{number of false neg.}}$$

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

$$\textbf{FP rate} = \frac{\text{number of false pos.}}{\text{number of false pos.} + \text{number of true neg.}}$$

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

$$\textbf{Efficiency} = \frac{\text{number of true pos.}}{\text{number of true pos.} + \text{number of false pos.}}$$

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,

	Dataset I				Dataset II			
	GTR	TRW		EM	GTR	TRW		EM
GTR Classification	count	scanner	benign	scanner	count	scanner	benign	scanner
Benign	309	0	2	62	1,380	0	0	114
Likely Benign	4,825	0	1	0	2,427	0	0	0
Scanner	366	343	0	366	5,44	174	0	544
Likely Scanner	201	65	0	196	758	9	0	750
Unknown (one failed)	799	0	0	797	24,200	0	0	23,810
Unknown (others)	127	1	2	9	1,784	5	1	749
Total	6,627	409	5	1,430	31,093	188	1	25,967

Table 3.5: The distribution of declared scanners by TRW and EM among the categories of GTR (dataset I of Jan 28 to Mar 10, 2007; dataset II of Jun 14 to Jul 4, 2010).

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., number of false negatives/(number of false negatives + number of true negatives)) is a more meaningful performance metric than TP rate. Based on the datasets studied in the literature (e.g., [6]) and our datasets, network scanning activity is often of the former case (i.e., when the number of true negatives are larger than true positives).

Table 3.5 shows the distribution of TRW declared scanners/benigns and EM detected scanners among the categories of our GTR. To avoid using a GTR which was data-fitted to the datasets it is itself evaluating, note that we removed three days from dataset I (Jan 28 to Mar 10, 2007 instead of Jan 28 to Mar 13, 2007) and added three days to dataset II (Jun 14 to Jul 4, 2010 instead of Jun 17 to Jul 4, 2010) in Table 3.5. In total, TRW declared 409 and 188 scanners, whereas EM declared 1,430 and 25,967 scanners in the first and second datasets, respectively. For the remotes in the scanner category, TRW declared 343 and 174 scanners, whereas EM detected 366 and 544 scanners in the first and second datasets, respectively. The sensitivity of EM (marking a remote as a scanner after the first inbound connection attempt to non-existing network service) also caused EM to detect more scanners than TRW for

those in the likely scanner category (65 and 9 detected by TRW versus 196 and 750 detected by EM in the first and second datasets, respectively).

While none of the remotes in the benign or likely benign categories (per the GTR) were declared by TRW as a scanner, EM declared 62 and 114 of the remotes in the benign category as scanners. According to the benign rule (see Table 3.3) and EM classification criterion (see Section 2.7), each remote in the benign rule that EM classified as scanner made failed connection attempts with at most one {IP_addr, dst_port} tuple not in the NEM table, while also either making successful connections with at least three distinct {IP_addr, dst_port} tuples or that at least one local host has initiated an outbound connection to the remote.

For the first unknown category (one failed), TRW did not flag any remote as scanner, as opposed to EM flagging the majority of them. This is because, as a function of the TRW default parameters (to the best of our knowledge, there is no publicly available criterion specifying how to choose appropriate values in TRW), TRW requires at least four consequent failed connection attempts initiated to four distinct local hosts within a given time window for a remote to be classified as a scanner, while in EM, a remote with failed connection attempts destined to one or more services not in the NEM table is flagged as a scanner. Both algorithms flagged some of the remotes in the second unknown category (others) as scanners.

To calculate the detection accuracy metrics, the GTR categories are merged into three derived GTRs (representing the possible reasonable binary combinations of the GTR categories to compare against the detectors' binary results of scanner/non-scanner) as follows:

GTR1) remote hosts in both the scanner and the likely scanner categories are true positives and the remainder are (i.e., the benign, likely benign, and the two unknown categories); are true negatives.

GTR2) true positives are only those in the scanner category and true negatives are those in the benign, likely benign, and unknown categories, while remotes in the likely scanner category are omitted entirely, i.e. they are included in neither the true positives nor the true negatives.

Performance Metrics		Dataset I		Dataset II	
		TRW	EM	TRW	EM
$GTR1$	TP rate	0.7196	0.9912	0.1405	0.9939
	FP rate	0.0002	0.1432	0.0002	0.8282
	Efficiency	0.9976	0.3930	0.9734	0.0498
$GTR2$	TP rate	0.9372	1.0000	0.3199	1.0000
	FP rate	0.0002	0.1432	0.0002	0.8282
	Efficiency	0.9971	0.2966	0.9721	0.0216
$GTR3$	TP rate	0.9372	1.0000	0.3199	1.0000
	FP rate	0.0002	0.0135	0.0009	0.1544
	Efficiency	0.9971	0.8375	0.9721	0.3866

Table 3.6: Evaluation of TRW and EM detection accuracy.

GTR3) true positives are only those in the scanner category and true negatives are those in the benign, likely benign, and unknown (others) category, while remotes in both likely scanner and the unknown (one failed) categories are omitted entirely, i.e. included in neither the true positives nor the true negatives. As discussed in Section 3.3.2, the remotes in the unknown (one failed) category are hard to classify correctly since while they made failed attempts with only one network service, which is not enough evidence to have confidence of malicious intent, they did not make any successful connections. Therefore, GTR3 takes the approach that it seems unfair to penalize the evaluated algorithms for classifying these remotes as either scanners or benign.

Table 3.6 (see also Figure 3.6) shows the performance of both algorithms in terms of detection accuracy according to the metrics discussed above using GTR1, GTR2, and GTR3. The following is a summary of the evaluation results.

GTR1. In both datasets, EM has a significantly better TP rate than TRW. However, this was at the expense of a high FP rate, especially in dataset II (EM FP rate = 0.83). The TRW TP rate was low in dataset II because there were many scanners that contacted less than the TRW default threshold number of network services for declaring remotes as scanners (i.e., less than four). The very high false positive rate with EM is because of the high number of remotes that made failed connection

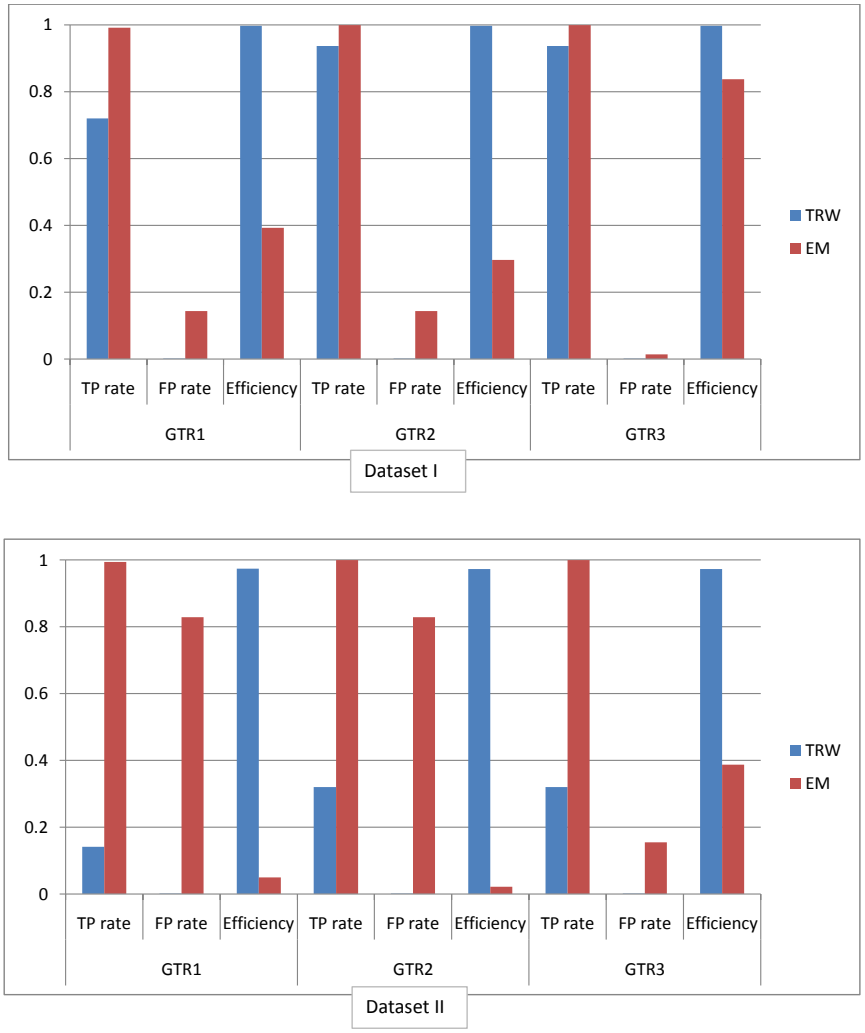


Figure 3.6: TRW and EM detection accuracy as in Table 3.6.

attempts with only one network service, which EM considers as scanners while GTR1 does not. Consequently, the efficiency of TRW is significantly better than EM.

GTR2. Recall the "likely scanner" category is omitted in GTR2. Consequently, the TP rate improved noticeably for TRW (an increase of about 30% in both datasets). This is because, unlike TRW, the "likely scanner" category captures scanners with low scanning rate (even remote hosts that unsuccessfully attempted to connect to only two distinct local addresses) and those for which most of their connection attempts fail

(i.e., with $R_\phi \geq 0.75$). FP rates for both EM and TRW remain similar to those with GTR1, as omitting the likely scanner category is not affecting both the number of false positives and true negatives. Although the TP rate increased for both algorithms, the number of true positives decreases since detected scanners that fall into the likely scanner category are not counted, unlike the number of false positives that remains the same. Accordingly, relative to GTR1, the efficiency decreased in both algorithms.

GTR3. GTR3 is similar to GTR2 except that the unknown (one failed) category is omitted. Therefore, TP rates are similar to those with GTR2. However, in both datasets, the EM FP rate improved dramatically, as EM flags each remote in the unknown (one failed) category as a scanner (which is omitted in GTR3), and thus EM efficiency also increased significantly.

3.4 Concluding Remarks

In Chapter 8, we give a comparative summary of several scanning detection algorithms including TRW and EM. In particular, Table 8.1 provides a comparative overview of these algorithms including detection accuracy, limitations and strengths, and evasion resistance. Section 8.2 discusses the network environments that fit each scan detection algorithm.

In this chapter, we provide a detailed analytical and empirical comparison of two known scan detection algorithms, TRW and EM, discussing features, capabilities, and limitations of both algorithms. We also study the impact of these algorithms' parameters in terms of computational resources and detection accuracy and speed.

We evaluate the detection accuracy of these algorithms by conducting several experiments on real-world network traces. The results show that the TRW false positive rate is significantly less than that of the EM algorithm, but this comes at the cost of the effectiveness (i.e., detection rate) of TRW. In order to establish a reference baseline of scanners to compare against in the studied network traces, we present a classification criteria in which network traffic of remote hosts are examined during the entire capture period of each trace, classifying each remote host into one of six categories according to a set of benign and scanning behaviour heuristics.

This chapter aids in understanding the strengths and limitations of the state of the art in detecting scanners. Chapters 4 and 5 leverage this understanding to enhance existing detection schemes and to develop a new, more efficient, feature-enhanced scan detection algorithm. Given that the TRW algorithm is now considered in the literature as the current state-of-the-art for scan detection (e.g., see [97]) and that the TRW's detection accuracy (FP rate and efficiency in particular) outperforms the EM algorithm (as in our evaluation in Section 3.3), we only compare our new scan detection algorithms in Chapters 4 and 5 with the TRW algorithm. In Chapter 8, we provide a comparative summary of TRW, EM, and two new scan detection algorithms (presented in Chapters 4 and 5).

Chapter 4

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

In this chapter, we leverage the gained observations and insights from Chapter 3 in designing and implementing a new scan detection algorithm. 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. 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. 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.

4.1 Introduction

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 chapter, 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 [45]. Unlike TRW, LQS can detect vertical scans and it has a greater resistance 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 attempt to the monitored network. Unlike previous scan detection approaches (e.g., [45, 90]), 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.

Our implementation of LQS (Section 4.3.2, Algorithm 3) as a policy in the Bro IDS [2] is given in Appendix A (also 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. Challenges in real-time scan detection are discussed in Section 4.2. We present a design overview of LQS in Section 4.3. Section 4.4 explores the advantages of LQS relative to TRW discussing the features and capabilities of both. Section 4.5 evaluates LQS on two datasets from different sites; scan detection results of both the LQS and TRW algorithms are given and analyzed. Section 4.6 concludes.

4.2 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 [20], 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.

Resistance 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.

4.3 LQS: Online Scan Detection Algorithm

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

4.3.1 Overview

The algorithm depends on failed connection attempts as an indication of network scanning activity as discussed in Section 3.3.2. The LQS algorithm uses the exposure maps technique [124] 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 3) 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_addr, dst_port} tuples. 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., [46]); and 2) quickly capturing a remote host change in state (i.e., being compromised).

4.3.2 Design Details

Pseudo-code of LQS is given in Algorithm 3. 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 {IP_addr, dst_port} tuple is added to active network services table OPS or the corresponding entry is refreshed if it is already in OPS (line 23). If the tuple 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 tuple. 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_IP_addr, dst_port} tuple contacted previously by the same remote host will not increase the remote host's counter. Only the first k unique {local_IP_addr, dst_port} tuples 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.

Algorithm 3: LQS (returns *True* when a new IP address is classified as a scanner)

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) tuples 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} tuples`[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) ∧ [C.dstIP, C.dstPORT] ∉ (OPS ∪ CPS) ∧ [C.dstIP, C.srcIP] ∉
           CR) then
4              if ([C.srcIP] ∉ FC) then add new entry for index C.srcIP into FC
5              if (FC[C.srcIP].count < k) then
6                  if (C.dstIP ∉ 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 ⇐ FC[C.srcIP].count + 1   //src,dst didn't contact previously
10                 else if (C.dstPORT ∉ 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 ⇐ 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) ∧ (C.dstIP ∉ S) ∧ (RST ∉ C.flags) then
20             add new entry for index C.srcIP, C.dstIP into CR
21         end
22     else if SuccessfulConnection(C) ∧ IsLocalAddress(C.dstIP) then
23         add [C.dstIP, C.dstPORT] to OPS
24         if [C.dstIP, C.dstPORT] ∈ CPS then
25             delete [C.dstIP, C.dstPORT] from CPS
26         end
27         if (C.dstPORT ∈ 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 ⇐ FC[C.srcIP].count − 0.25
31             else
32                 delete FC[C.srcIP].Contacted[C.dstIP]
33                 FC[C.srcIP].count ⇐ 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)) ∧ ([C.dstIP, dstPORT] ∈ OPS) ∧ 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
```

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 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).

4.3.3 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

[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 tuples 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.

connection attempts with local network services (i.e., contacting more {local_IP_addr, dst_port} tuples) 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 tuples within I_3 time window will not be reported. Given that a connection attempt destined to a tuple 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.

4.3.4 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 an 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).

4.4 Advantages over TRW

This section illustrates the advantages of LQS over the TRW algorithm [45]. 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 4.5).

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 {local_IP_addr, dst_port} tuple is in neither the OPS nor CPS tables.

Schechter et al. [95] 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. [44] 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.

Resistance 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 [46]. 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 3.3.1). 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 tuples a remote host unsuccessfully attempted to contact where the list is incremented only if the remote unsuccessfully contacts a new tuple. 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. 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} tuples 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.

4.5 Empirical Evaluation

In this Section, we first describe our experimental setup including the datasets and how to establish a ground truth reference (GTR) of scanners, and then present the results of our experiments on the LQS algorithm and compare them with TRW results (see Section 3.3.3 for TRW empirical evaluation).

4.5.1 Datasets, GTR, and Setup

We used the two datasets described in Section 3.3.1 (Jan 28 to Mar 13, 2007 and Jun 17 to Jul 4, 2010 for datasets I and II, respectively). To establish a ground truth reference (GTR) of scanners, we used the same methodology described in Section 3.3.2.

As in Section 3.3.2, the GTR classification criteria consists of six rules: *benign*, *likely benign*, *scanner*, *likely scanner*, *unknown-one* (for those contacted unsuccessfully by only one host), and *unknown* (for the rest). The rules are matched in order from the *benign* class to *unknown*, where a remote host can only be assigned to one class at a given time.

The criteria is based on the state of inbound connection attempts and outbound connections using the following parameters: (1) the number of distinct IP address and destination port tuples a remote host initiated successful connection attempts to; (2) the number of distinct tuples the remote initiated failed connection attempts to; (3) the number of distinct tuples that initiated connection attempts (whether successful or failed) to the remote. The distribution of the detected scanners by TRW and EM among the categories of GTR is given in Table 4.1.

We have implemented LQS (see Appendix A) in the Bro language (Bro 1.4 NIDS [2]) 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. We use the detection accuracy metrics used in Section 3.3.2.

4.5.2 Results

Table 4.1 shows the distribution of TRW and LQS detected scanners among the categories of our GTR. 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 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 GTR.

GTR Classification	Dataset I			Dataset II		
	GTR count	TRW	LQS	GTR 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 4.1: The distribution of the detected scanners by TRW and LQS among the categories of GTR (dataset I of Jan 28 to Mar 13, 2007; dataset II of Jun 17 to Jul 4, 2010).

Performance Metrics		Dataset I		Dataset II	
		TRW	LQS	TRW	LQS
GTR1	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
GTR2	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 4.2: Detection accuracy results.

To calculate the detection accuracy metrics, the GTR categories are merged into two derived GTRs as follows: (GTR1) remote hosts in both the *scanner* and the *likely scanner* classes are considered true positives, whereas all other classes are considered true negatives; and (GTR2) remotes in the *scanner* class are considered true positives, whereas those in the *benign*, *likely benign*, and *unknown* are considered true negatives (remotes in the *likely scanner* class are omitted).

Table 4.2 shows the detection accuracy results of both algorithms according to the metrics discussed in Section 3.3.2 using both GTR1 and GTR2. The results are summarized as follows.

Dataset I and GTR1.. 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 GTR2.. 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 3.3.2).

Dataset 2 and GTR1.. 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 GTR2.. Even with GTR2, 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. [6]), 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 [59, 60]).

4.6 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 [35] 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 resistance to evasion. We also presented a novel methodology to obtain an estimated ground truth for evaluating network scanning detectors.

Note that the two datasets used in this chapter to compare the detection accuracy of TRW with that of LQS are also used in Chapter 3 to compare the detection accuracy of TRW and EM. Therefore, a direct comparison of the detection accuracy of LQS and EM can be inferred by comparing Table 4.2 in this chapter with Table 3.6 in Chapter 3. In Chapter 8, we provide a comparative summary of TRW, EM, LQS, and STRW (a proposal of a new scan detection algorithms presented in Chapter 5).

Chapter 5

Revisiting Network Scanning Detection Using Sequential Hypothesis Testing

As discussed in Chapters 3 and 4, behaviour-based scanning detection techniques based on the state of inbound connection attempts remain effective against evasion, as the objective of network scanning is to find responsive services and thus it is assumed that the adversary does not know the open ports in the monitored network. Many of today's network environments, however, feature a dynamic and transient nature with several network hosts and services added or stopped (either permanently or temporarily) over time. In this chapter, working with recent network traces from two network environments (different than the two datasets used in Chapters 3 and 4), we re-examine the TRW (Threshold Random Walk) scan detection algorithm and we show that the number of false positives is proportional to the transiency of the offered services. To address the limitations found, we present a modified algorithm (STRW) that utilizes active mapping of network services to take into account benign causes of failed connection attempts. STRW eliminates a significant portion of TRW false positives (e.g., 29% and 77% in two datasets studied).

5.1 Introduction and Motivation

Most scan detection techniques are designed for deployment in an enterprise environment and a controlled environment is often assumed where only limited network services are allowed to operate usually through continuously running servers (e.g., web, mail, and DNS servers). Scan detection schemes based on a remote host's successful or failed connection attempts are ideal for such environments because a failed connection is a good indicator that the remote host lacks knowledge of the available services and thus might be a scanner. That is, assuming a stable network where services and server IP addresses are rarely changed, the probability that benign remotes

make failed connections is low. In this case, causes of failed connection attempts from benign sources are usually either misconfigured remote hosts (e.g., setting a wrong destination IP address or port for a service) or some special cases, e.g., web crawlers and proxies (many users could be represented by a single proxy IP address and their traffic through the proxy may look like scanning activity).

On the other hand, in an enterprise with a more relaxed security policy and fewer restrictions on what applications or network services can be used by enterprise workstations (or sometimes with no explicit security policy; e.g., some university networks), new hosts may be added either for the short or long term, host mobility is possible, and there are less restrictions on the client applications or services run by hosts. In enterprise networks which now commonly enable wireless access, the availability of hosts offering services may change rapidly with the device states of the host machines (on/off, hibernated, or disconnected). Furthermore, with dynamic IP configuration, different IP addresses can be assigned to the same physical network port over time. Therefore, transient network services may appear and disappear as devices are powered on/off or physically removed (e.g., in the case of services hosted on mobile devices like laptops). Even in a controlled enterprise environment, some network services might be temporarily unavailable due to maintenance in the hosting servers or network failures.

In many of today's network environments such as these, many benign remote hosts may make failed connection attempts because the services they are trying to connect to, while active in the past, are temporarily unavailable or disabled (because the local host running the service is turned off/sleeping/hibernating, disconnected from the network, or the application running the network service is uninstalled or closed). Thus, if relying on failed connections as evidence of malicious intent in such environments, these issues need to be taken into account for designing a practical network scan detection algorithm, to reduce significantly false positives.

In this chapter, we study the performance of the TRW scanning detection algorithm (see Section 2.6 for background on TRW), which is based on a remote host's successful or failed connection attempts, in such network environments. Based on experimental results from a set of tests on traces from two network environments,

we analyse the traffic of remote sources flagged as scanners by TRW showing that a significant number of these sources are false positives (19% - 50%) exhibiting benign behaviour rather than scanning. To overcome this drawback, a new algorithm (STRW) introducing modifications to TRW is proposed which decreases significantly the identified false positives (on average 77% and 29% of TRW false positives are avoided in two datasets collected at different sites). STRW uses network services active mapping method that is used in the LQS scan detection algorithm (see Chapter 4 for background on LQS) as a decision oracle while maintaining the method of sequential hypothesis testing used in TRW (see Section 5.2.2 for a comparison between STRW and LQS).

Contributions.

1. STRW: We confirm that TRW was designed for scan detection in a controlled enterprise network environment, identifying several causes of false positives in now common environments of transient nature. Accordingly, we propose a modified algorithm (STRW) for scan detection which takes into account the identified causes of TRW false positives in such environments. STRW shows that the believed hypothesis that behaviour-based network scanning detectors (e.g., TRW) exhibit unsatisfactory performance in residential style network traffic [103] is due to the lack of utilizing information of the characteristics of the monitored environment. In particular, utilizing the monitored network profile to identify benign causes of unsuccessful connection attempts improves significantly the performance of TRW in such environments. We believe that the contribution should be weighed not only by the magnitude of the changes to the original TRW algorithm, but their effect on the algorithm performance in terms of the number of false positives.

2. Empirical Evaluation: We implement STRW (Section 5.2, Algorithm 5) as a policy in the Bro IDS [2] (See Appendix B; the implementation is also available at http://strw.sourceforge.net/). We evaluate the detection accuracy of both TRW and STRW on class C and class B networks.

3. Obtaining Ground Truth of False Positives: We propose using the exposure maps technique [124] as a proxy for ground truth for identifying a lower bound on scan detection false positive results.

Organization. We propose and evaluate STRW, a modified TRW algorithm, in Section 5.2. STRW advantages and limitations relative to other scan detection algorithms are given in Section 5.2.2. The used datasets and their network environments are given in Section 5.3.1. We present our evaluation methodology in Section 5.3. Results and analysis of testing TRW and STRW on each dataset are given in Section 5.4. Section 5.5 concludes the chapter.

5.2 Stateful TRW (STRW)

In this section, we propose modifications (see Algorithm 5) to the TRW algorithm in a new algorithm that we call *Stateful TRW* (STRW). STRW represents a hybrid approach that uses the LQS decision oracle to determine whether a new connection attempt is potentially malicious while maintaining the method of sequential hypothesis testing used in TRW to classify remotes as either scanner or benign. This section also shows the STRW advantages and limitations relative to TRW and LQS algorithms.

5.2.1 Algorithm

Unlike the two tables in LQS, STRW has a table of network services, NS, for keeping the state of open ports in the monitored network as in lines 8 to 10. For convenience and easier comparison, we reprinted Algorithm 1 in Algorithm 4 which is next to Algorithm 5 (for an explanation of the use of expiry intervals and other keywords, see Section 2.6.2). The IF statement in lines 11-17 in TRW (Algorithm 4) is replaced by lines 14-20 in the new algorithm (Algorithm 5). First, if the connection is successful and the remote host did not make a successful connection with the local host before (i.e., source/destination IP addresses pair is not in the SC set), the remote host's likelihood ratio is lowered towards being classified as benign. Otherwise, if the targeted network service is not in NS (which indicates implicitly that the current connection attempt is unsuccessful) and the remote host did not make a failed connection attempt

Algorithm 4: TRW (returns True when a new IP is classified as a scanner).

INPUT:
β(def:=0.99), α(def:=0.01), θ_0(def:=0.8), θ_1(def:=0.2)
I_1(def:=1hr), I_2(def:=1hr), I_3(def:=0.5hr), I_4(def:=0.5hr), I_5(def:=0.5hr), I_6(def:=0.5hr)
C `//data structure holding current connection information.`

OUTPUT:
S (def:=$\emptyset$, expires after I_1) `//set of scanners' IP addresses.`
B (def:=$\emptyset$, expires after I_2) `//set of benign IP addresses.`
FC (def:=$\emptyset$, expires after I_3) `//set of IP address pairs with failed connection.`
SC (def:=$\emptyset$, expires after I_4) `//set of IP address pairs with successful connection.`
R (def:=$\emptyset$, expires after I_5) `//set of friendly remote IP addresses.`
L (expires after I_6) `//table of likelihood ratios of remote hosts (Λ).`

```
1  begin
2    if IsLocalAddress(C.srcIP) then
3      if C.dstIP ∉ S then
4        add C.dstIP to R
5      end
6      return (False) //since it is outbound connection
7    end
8    if C.srcIP ∈ (S ∪ B ∪ R) then
9      return (False) //remote is already flagged as scanner, benign, or friendly.
10   end
11   if FailedConn(C) ∧ ([C.srcIP, C.dstIP] ∉ FC) then
12     add [C.srcIP, C.dstIP] to FC
13     ratio ⇐ (1 − θ1)/(1 − θ0)
14   else if SuccessfulConn(C) ∧ ([C.srcIP, C.dstIP] ∉ SC) then
15     add [C.srcIP, C.dstIP] to SC
16     ratio ⇐ θ1/θ0
17   else return (False)
18   if (an entry in L already exists for C.srcIP) then
19     L[C.srcIP] ⇐ L[C.srcIP] ∗ ratio
20   else
21     add new entry for index C.srcIP into L
22     L[C.srcIP] ⇐ ratio
23   end
24   if L[C.srcIP] > (β/α) then
25     add C.srcIP to S
26     return (True)
27   else if L[C.srcIP] < ((1 − β)/(1 − α)) then
28     add C.srcIP to B
29   end
30 end
```

with the local host before (i.e., source/destination IP addresses pair is not in the FC set), then the remote host's likelihood ratio will be raised towards being classified as a scanner. Therefore, in the new algorithm, failed connection attempts to previously

Algorithm 5: STRW (returns true when a new IP is classified as a scanner)

INPUT: `//for an explanation of the use of expiry intervals, see Section 2.6.2`
β(def=0.99), α(def=0.01), θ_0(def=0.8), θ_1(def=0.2)
I_1(def=1hr), I_2(def=1hr), I_3(def=0.5hr), I_4(def=0.5hr),
I_5(def=0.5hr), I_6(def=0.5hr), I_7(def=72hr)
C `//data structure holding current connection information`

OUTPUT:
S (def=$\emptyset$, expires after I_1) `//set of scanners' IP addresses`
B (def=$\emptyset$, expires after I_2) `//set of benign IP addresses`
FC (def=$\emptyset$, expires after I_3) `//set of IP addresses pairs with failed connection`
SC (def=$\emptyset$, expires after I_4) `//set of IP addresses pairs with successful connection`
R (def=$\emptyset$, expires after I_5) `//set of friendly remote IP addresses`
L (expires after I_6) `//table of likelihood ratios of remotes (i.e., Λ)`
NS (def=$\emptyset$, expires after I_7) `//set of offered network services (IP, port)`

begin
  if *IsLocalAddress(C.srcIP)* **then**
    if $C.dstIP \notin S$ **then**
      add $C.dstIP$ to R `//a local host initiated a connection first to dstIP`
    end
    return (False) `//since it is outbound connection`
  end
  if *SuccessfulConn(C)* **then** `// SYN-ACK packet is sent by the destination`
    add $[C.dstIP, C.dstPORT]$ to NS
  end
  if $C.srcIP \in (S \cup B \cup R)$ **then**
    return (False) `//a remote is already flagged as scanner, benign, or friendly`
  end
  if *SuccessfulConn(C)* $\wedge$ $([C.srcIP, C.dstIP] \notin SC)$ **then**
    add $[C.srcIP, C.dstIP]$ to SC
    ratio $\Leftarrow \theta_1/\theta_0$
  else if $([C.dstIP, C.dstPORT] \notin NS) \wedge ([C.srcIP, C.dstIP] \notin FC)$ **then**
    add $[C.srcIP, C.dstIP]$ to FC
    ratio $\Leftarrow (1-\theta_1)/(1-\theta_0)$
  else return (False)
  if *(an entry in L already exists for C.srcIP)* **then**
    $L[C.srcIP] \Leftarrow L[C.srcIP] * ratio$
  else
    add new entry for index $C.srcIP$ into L
    $L[C.srcIP] \Leftarrow ratio$
  end
  if $L[C.srcIP] > (\beta/\alpha)$ **then**
    add $C.srcIP$ to S
    return (True)
  else if $L[C.srcIP] < ((1-\beta)/(1-\alpha))$ **then**
    add $C.srcIP$ to B
  end
end

offered services are now ignored. Consequently, a benign remote host that happens to attempt connecting to any of the network services that are temporary unavailable will not get penalized by TRW for making such failed connection attempts since its likelihood ratio will not be updated.

Entries in NS expire after interval I_7 so that if there is no traffic from the port offering the service indicating the port is still open (e.g., SYN-ACK packet in case of TCP protocol) for a period of I_7, the NS entry will be removed. Choosing an appropriate value for I_7 depends on the properties of the monitored network and the type of the offered network services. I_7 should reflect the approximate duration of inactivity after which a network service is most likely being removed permanently from the monitored network. Ideally, each offered service might have a different expiration time. However, one common parameter for all services is easier to set by administrators given that network services might run on non-standard ports. Using a long expiration time increases false negatives since if a network service is not offered anymore and I_7 is not yet expired, all failed connection attempts to this service will be ignored by STRW. On the other hand, using a short expiration time might increase the number of false positives since failed connection attempts to the service after I_7 expires while the service is temporarily unavailable, will trigger STRW to adjust the corresponding Λ_r towards a scanner.

5.2.2 Advantages and Limitations Relative to TRW and LQS

Relative to TRW, STRW addresses many of the causes of failed connection attempts from benign sources and therefore it has better detection accuracy in terms of false positives. Unlike TRW (see Section 2.6), STRW is designed for network environments of transient nature (e.g., wireless and residential usage patterns). While TRW has to wait for the connection state to be known (i.e., whether it is successful or unsuccessful), which gives adversaries a time window to perform further scanning before being detected (see Section 5.2.2), STRW uses the NS table to immediately decide if the connection attempt is unsuccessful. Thus, STRW is more appropriate for scanning worm detection and it has a better resistance to evasion. Failed connection attempts determined by STRW indicate malicious intent more than those in TRW

and thus STRW default parameters can be tightened towards faster detection (see Section 5.2.3).

Unlike LQS (see Chapter 4), the sequential hypothesis testing model in STRW provides a theoretical basis for classifying remote hosts. Also, TRW credits remotes that make successful connections by reducing their likelihood ratio towards being classified as benign. While the objective is to avoid penalizing benign hosts that make some failed connection attempts, it enables scanners with a priori knowledge of some available services of the target network to delay detection. On the other hand, LQS can detect vertical scans and it requires a smaller memory footprint than STRW.

STRW is similar to TRW and LQS in terms of being subject to evasion from a scanner with access to a large number of remote hosts (e.g., a botnet) since the scanner can divide the target IP range among these hosts so that each host can scan fewer IP addresses than the STRW threshold to avoid detection (other TRW limitations are discussed by Jung et al. [45]). Relative to TRW and LQS, a training period if needed (see Section 5.4) represents a limitation in STRW.

STRW should perform better than TRW at the ISP level since externally-accessible network services offered by home users are more likely to be transient (e.g., due to wireless connectivity or home devices being powered on/off). However, bi-directional flows with packet headers must be available for the detector; otherwise, false positives will be generated. In case only flow level traffic collection (e.g., NetFlow) is available, any scan detection mechanism that relies on packet headers will not be applicable (e.g., TRW or STRW). Such limitation could be due to the expected large required storage and computation resources to access packet headers at the ISP level.

Users accessing the Internet wirelessly (e.g., through mobile devices) are more susceptible to lose network connectivity and thus more likely to generate false positives with TRW if any application that requires opening a network port (e.g., some P2P clients and active mode FTP connection) is running in the user device. Therefore, STRW is also suitable for ISPs that provide mobile broadband services (e.g., using HSPA, WCDMA, WiMax and LTE technologies).

In Chapter 8, we give a comparative summary of the TRW, EM, STRW, and LQS scanning detection algorithms, including detection accuracy, essential properties,

limitations, and evasion resistance. In fact, had we presented the algorithms in their chronological order of conception, we would have first presented STRW (i.e., after Chapter 3 which compares the TRW and EM algorithms) and then LQS (Chapter 4). Indeed, STRW is a modified TRW algorithm and we do consider LQS to be a stronger contribution, which also suggests that it would have been better presented later in the thesis than STRW. However, we instead chose the current order because the evaluation methodology and the two datasets used in Chapters 3 and 4 are similar and are different than those used in this chapter.

5.2.3 Parameterization

While STRW's detection rate is similar to that obtained by TRW, STRW resulted in a substantially lower FD rate across various parameters values. As discussed earlier, choosing I_7 appropriately depends on the nature of the monitored network. However, given it is usually hard to understand the network behaviour which could also change over time, we recommend setting a large value for I_7 (e.g., $>$ 1 week). The possible increase in the FN number (compared to TRW) is expected to be insignificant because of the very low ratio of open ports to closed ports in all networks. Even with large I_7, the number of entries in NS is limited to the number of offered network services during this period which is expected to be much smaller than the size of the FC or the SC sets for the same period.

We recommend setting $I_3 = I_4 = I_6$ as this helps in decreasing the false positive rate and makes the effect of successful and unsuccessful connection attempts on Λ consistent with θ_0 and θ_1 setting (see Section 5.4). It is also desirable to set this value higher than the default (e.g., one day vs. 30 min default value) since a higher value detects more stealthy scanners. A possible improvement to STRW is to set I_6 greater than I_1 and I_2 such that, for a remote host r, the corresponding Λ_r continues to be updated even if r is already classified (S and B must be removed from the condition in line 11). If r is already in B and Λ_r exceeds the upper threshold (line 27), r should be first removed from B. Similarly, if r is already in S and Λ_r crosses the lower threshold (line 30), r should be first removed from S.

Number of:	Inbound	Outbound
a) Flows (TCP, UDP, and ICMP)	20,238,134	64,956,098
b) TCP connections (flows)	2,285,486	38,619,016
i) Successful TCP connections[1]	61%	9%
ii) Rejected TCP connections[2]	5%	3%
iii) Timed-out TCP connections[3]	34%	88%
c) Remotes initiating TCP connections	478,684	156

Table 5.1: Dataset statistics of Class C network (dataset of March 15-28, 2009; (i), (ii), and (iii) are percentage of b)

Since STRW is designed to reduce false positives, we recommend raising the default value for α (e.g., to between 0.05 and 0.10 where the original default was 0.01) to enable detection of more scanners and to flag more benign remote hosts. Setting θ_1 to a lower value than the default, and according to the density of the offered services in the monitored network (i.e., the ratio of open ports to closed ports of all Internet-addressable local hosts) will also improve accuracy. In our conducted experiments, reducing the default θ_1 value reduced slightly the FD rate.

5.3 Datasets and Evaluation Methodology

This section first describes the datasets and the associated network environments. It then describes our methodology in obtaining approximated ground truth for identifying false positives in TRW and STRW detection results.

5.3.1 Datasets and Network Environments

Class C dataset. We used a network trace of packet headers collected at two class C subnets of a university with 156 Internet-addressable IP addresses in total (see Table 5.1). Both subnets are located on the same network switch (primarily client network with no DNS server) where we gathered the trace over the period March

[1]SYN-ACK packet is sent by the destination.

[2]RST packet is sent by the destination after receiving SYN packet from the source.

[3]SYN packet or midstream traffic is sent by the source but no SYN-ACK packet is sent by the destination for at least 5 minutes.

12-28, 2009. We used the dataset for March 15-28 for all the experiments except the second set in Section 5.2 where we used the full trace. The size of the March 15-28 dataset is 381 gigabytes (on average 27 GB/day); 0.06% of the packets were dropped by the tracing program. All the IP addresses were populated (originated outbound traffic during the period).

The network has an open security policy in both subnets, with no restrictions on the network services the clients can offer. Although the assigned IP address of each network access point did not change during the capture period, each user can set up a router at their network access point to connect multiple machines (though most of the users did not have routers and thus only few NATs exist). For simplicity, we only study TCP traffic. We conducted the experiments off-line using the Bro 1.4 NIDS [2].

Using a signature-based detection approach (which is not central to the contributions in this chapter), Table 5.2 gives a summary of the observed protocols running in the open TCP ports in the Class C dataset. Since we collected only the maximum TCP/IP header size (the first 96 bytes of the packet), only protocol signatures located in the first bytes of the TCP payload data for packets with shorter than maximum header size are identified. Note that this limits a detailed network behaviour analysis. We do not rely on the destination port number to identify the corresponding protocol. Only the first initiated TCP connection to each of these open ports is analyzed and thus it is possible that other protocols not captured in Table 5.2 are used later through these open ports. Moreover, various protocols can be tunneled over some of the reported protocols in Table 5.2 (e.g., over HTTP or SSL). For instance, similar to web traffic, some protocols including P2P use HTTP requests for file transfer [47]. Table 5.2 indicates that the majority of the open ports in this dataset belong to what appear as network services used by MSN, HTTP, and P2P data transfer protocols.

Class B dataset. The dataset contains packet headers (554 gigabytes) and it is gathered from a class B university network (a university different than the one in Section 5.3.1) over the period February 1-11, 2010. The network environment is controlled in that the majority of the users can not install any software. No software that offers a network service is installed in any workstation by default; however, some

workstations have open ports (45% of TCP outbound connections failed vs. 95% of inbound connections failed). In both datasets, IP addresses are not shuffled between access points (e.g., by a DHCP server) and there is no DNS servers.

For the Class B dataset, the extracted network services table contains 498 open TCP ports. The open ports fall into the following categories: (a) 184 are web servers offering both http and https (most are printers offering web interface for configuration); (b) 68 FTP sessions (c) 46 SSH servers; (d) 4 DNS servers; (e) 2 SMTP servers; and (f) 185 various other services mostly on ephemeral ports.

5.3.2 Methodology

In this section, we first briefly review related work on the acquisition of ground truth of scanners in a given dataset, and then present our evaluation methodology for the TRW and STRW algorithms, including a new mechanism to establish ground truth of false positives.

Ground Truth of Scanners (Background)

Acquisition of ground truth data is needed to establish confidence in the results of any scan detection algorithm; the ground truth is an authoritative reference for determining TP and FP rates. While labeled datasets are typically used to evaluate the performance of an IDS technique, labeling a dataset with scanning activities is a far less scientific proposition, as is not conventional since intrusion signatures can not reliably assure the scanning intent. Also, generating synthetic scanning activity does not reflect the various possible port scanning mechanisms.

While it is possible to use IDSs to identify some scanners with network traffic containing known signatures for worms or network attacks, it is not always the case that network traffic originated from a scanner will contain malicious payloads. Carrying a malicious payload in the first packet of the connection is more likely when adversaries scan UDP ports while a scanner targeting TCP ports that can not make a successful connection (i.e., could not find an open port) is expected to send only SYN packets (e.g., an HTTP scanning worm). Therefore, IP addresses of sources sending malicious code might represent only an insignificant portion of the number

Protocol	Ports count	Description
MSN	2,826	MSN Messenger file transfer, audio, and video protocols
BitTorrent	114	P2P file sharing protocol
HTTP	137	Hypertext Transfer Protocol
NBSS	68	NetBIOS Session Service
SMB	43	Server Message Block (file or print sharing)
DCE/RPC	37	Distributed Computing Environment/Remote Procedure Calls
VoIP	12	Only Skype protocol is detected
SSL	10	Secure Sockets Layer
Others (known)	111	GTP, NDPS, TDS, UCP, OPSI, eDonkey, RMI, SLSK, COPS, DAAP, DIS, DISTCC, ENTTEC, Etheric, giFT, H1, LDAP, MSMMS, PCAP, PPTP, TNS, YMSG, Zebra, RTSP, CAST, Diameter, IDAP, X.25
Others (unknown)	1,603	Most packet payloads in binary format
Total	4,961	

Table 5.2: Protocols used in the offered services in the class C dataset.

of all scanners contacting the monitored network.

A possible way to get a measure of ground truth is to monitor (in a type of baseline training) remote hosts' network traffic over moderate periods of time (e.g., a few days) and flag those in which the majority of their flows are unsuccessful. For example, as a way to identify scanners Jung et al. [45] used the observation that for a given remote host (in their datasets), the percentage of the local hosts for which the connection attempt with the remote host failed is either very close to 0% or 100%. They then set a threshold of 80% or more failed connection attempts to identify scanners.

While this heuristic might be true for a large portion of scanners, it could also include a significant number of benign sources making repeated unsuccessful connections because of the unavailability of the contacted services as discussed in Section 5.1. Moreover, stealthy scanners who contacted some legitimately offered services in the monitored network might not exceed this threshold even over a relatively long period of monitoring and hence will not be flagged as possible scanners. On the other hand, it is hard to conclude with certainty that a source with only a single or very few connection attempts, the majority of them unsuccessful, is a scanner. Although there is no 100% error-free scan detection technique in the literature to date, comparing the detection results to other algorithms' results can also help in validating scan detection results.

Evaluation Methodology

The traditional metrics for measuring the performance of a scan detection algorithm are the true positive (TP) rate and false positive (FP) rate. The TP rate is the proportion of the distinct IP addresses of scanners that were correctly reported as scanners (TP rate $= TP/(TP + FN)$, where FN is the number of false negatives; i.e., distinct IP addresses of scanners that were erroneously classified as benign). The FP rate is the proportion of the distinct IP addresses of benign sources that were erroneously classified as being scanners (FP rate $= FP/(FP + TN)$, where TN is the number of true negatives; i.e., distinct benign source IP addresses that were correctly classified as non-scanners). Notice that a remote host is counted only once in computing TP, FN, FP and TN regardless of the number of connection attempts initiated by the remote.

Herein we propose using the EM technique to evaluate scan detection performance only from the perspective of number of discovered false positives. That is, we use the EM technique to identify most of the benign remote hosts and thus if TRW classifies any of them as a scanner it is counted as a false positive. In particular, we give an approximate measure of the false discovery rate (FD rate $= FP/(FP+TP)$) for the TRW algorithm, i.e., the proportion of the reported positive by TRW that are false positives.

Given that, regardless of the detector performance, the FP rate (FP rate $= FP/(FP + TN)$) of a scan detector is expected to be small (e.g., less than 1%) since the number of non-scanner remotes (i.e., TN) is significantly larger than the number of scanners (i.e., TP) in a given dataset (as discussed in Section 3.3.3), we argue that the FP rate is not a significant metric in determining the detection accuracy of scan detectors. Rather, the count of false positives and FD rate are better measures of the scan detector accuracy and the volume of false alarms in a given time window.

As discussed in Section 5.2, the modifications made in STRW aim to reduce the proportion of the actual negatives that are erroneously reported as positive by TRW and not to increase the number of hits (i.e., true positives). Therefore, ground truth of false negatives is not necessary. We choose not to use the proposed method to establish

an estimate ground truth of scanners in Section 3.3.2 as our proposed method gives a more solid ground truth for identifying false positives, which is what we require for the evaluation.

We have implemented the EM technique in the Bro language for validating TRW results. First, we used EM to enumerate network services offered by the local hosts in the dataset (as described in Section 2.7) and also the services availability based on timestamps. Knowing previously all actively responding ports in the network (i.e., by creating the NEM table as in Section 2.7), we traverse the dataset flagging any remote host initiating a TCP connection to a local IP/port not in the offered service table as a possible scanner; we call this list of remotes A.

While the list might contain benign sources, it will contain all scanners excluding those who probed only open ports. However, given that a network service in the NEM table might not be offered before being added to the table, it is possible that some remote hosts making failed connection attempts to this service before being added to NEM are actually scanners. Nevertheless, the probability $Pr(FN)$ of a specific scanning IP address coincidently attempting connections with only the network services in the NEM table is very low; $Pr(FN) = \prod_{i=1}^{n} \frac{no.\ of\ entries\ in\ NEM}{2^{16} \times\ no.\ local\ hosts}$ where n is the number of connection attempts from the remote host (assuming that all ports are scanned with equal probability). Therefore, remote hosts not in the list A that are flagged as scanners by a scan detection mechanism are false positives with a very high probability.

Using the default values of TRW parameters as in the Bro 1.4 implementation [2], any remote host IP address in the scanner's set will be removed as a member of the set after a one hour time period even if there is scanning activity from the same remote host within this hour, and likewise for the sources in the benign set. From a preliminary experiment using the default parameter values in the Bro implementation (see Algorithm 4), it was not clear how to deal with a continually changing list of scanners. Specifically, after observing all the IP addresses added to the scanner list over time and likewise for the benign list (sets S and B in Algorithm 4), about one third of the IP addresses in the scanner list also existed in the benign sources list which suggests that about one third of the marked remote hosts exhibit both malicious and benign behaviors. While it is possible that some remote hosts might

change state from benign to a scanner (e.g., compromised by a worm) or vice versa, many of these remote hosts were marked by TRW as benign and then scanner (or the opposite) several times during the network trace time frame.

To overcome this issue, we made both lists permanent as in the original TRW implementation [43] (i.e., expiration times I_1 and I_2 are removed). We also set expiration time I_6 to one day rather than the default value (30 minutes) to be able to detect more stealthy scanners (i.e., those which do not exceed TRW threshold within the default time window). However, TRW requires much more memory (e.g., more than 1GB of RAM with the particular dataset used) if all the write-expiry intervals in Algorithm 4 (i.e., $I_1 - I_6$ where the element is deleted when the given amount of time has lapsed since the last time the element was inserted in the set or the table) are removed as in the original TRW implementation. Therefore, the Bro implementation [2] with the default write-expiry intervals is more memory-efficient, but decreases detection accuracy.

We then performed several experiments using different values for TRW parameters. In each experiment we compare the list S of scanners flagged by TRW to the possible scanners list A we created earlier. It is expected that TRW will detect a subset of A, however, IP addresses in S and not in A represent remote hosts possibly misclassified as scanners by TRW. These IP addresses made connection attempts to only IP/port tuples that exist in the NEM table, and thus it is unlikely that they were performing network scanning. Therefore, these IP addresses represent the subset of TRW detected scanners that are false positives. We call this set TRW approximated ground truth aGT_{FP}. If at least one of the failed connection attempts that TRW considers in classifying a remote host as a scanner is destined to an IP/port tuple in the NEM table, this is also a possible false positive. We call this set $TRW\ aGT^*_{FP}$ ($aGT_{FP} \subseteq aGT^*_{FP}$). For the experiments in Section 5.3 we use $TRW aGT_{FP}$ set since the probability of having a scanner in this set is lower than for $TRW\ aGT^*_{FP}$.

5.4 Evaluation

This section reports the results of testing TRW and STRW on both datasets. False discovery rate ($FP/(FP+TP)$) is given for each experiment.

5.4.1 Class C Dataset

TRW. The NEM table of the network trace contains 4961 TCP ports offered by 78 unique local IP addresses. While there are 4250 distinct open TCP ports, there are only 5 non-ephemeral ports (80, 139, 443, 445, 1024) having only 19 distinct entries in the network offered services table. The most frequent open port is port 80 with 13 entries and then port 1412 with 6 entries. Over 60% of the table entries are from the dynamic or private ports (i.e., from 49152 through 65535). While there are 19 hosts with over 100 open ports for each, the maximum number of open ports per host over the network traffic capture period is 627 and the average number is 32 per host. A local IP address with a high number of open ports is either used by one machine running network services that use different ports at different times, or assigned to different machines during the capture period. $Pr(FN) = \prod_{i=1}^{n} \frac{4961}{2^{16} \times 156}$ where n is the number of connection attempts from the remote host (e.g., $Pr = 0.0005$ for $n = 1$).

Table 5.3 lists the results of ten selected TRW experiments with values for TRW parameters as shown (default input parameter values of Algorithm 4 are used except as noted). The number of false positives represents the size of $TRWaGT_{FP}$ (as defined in Section 5.3.2). The first observation is the dramatic change in the number of detected scanners and benign sources with only minor change to one of the TRW parameters. For example, the number of detected scanners triple by changing the α from the default value, 0.01, to 0.10. The parameters θ_0 and θ_1 affect how a remote host's likelihood ratio gets updated. α and β affect the upper and lower thresholds for flagging a remote host as either scanner or benign. The value of β (the desired detection probability) is typically set very high since the objective is to detect the scanners [45]. However, while it is always desired to have a lower false positive rate, increasing α increases the number of detected scanners as well.

To the best of our knowledge, there is no publicly available criterion specifying how to choose appropriate values in TRW to get satisfactory detection results. In addition, the TRW implementation (as in the Bro IDS [2]) contains 'write expire' intervals for the sets S, B, FC, SC, R and the table Λ (as in Algorithm 4) so that an entry is deleted from the set/table if not inserted again within a predefined time window. Such intervals, while helpful to avoid running out of memory especially

Experiment number	Non-default parameters - I_1 and I_2 are not used - $I_6 = 24hr$	no. of benign remotes	no. of remote scanners	no. of FP (distinct scanners)	FD rate (distinct scanners)
1	None	9,734	5,071	2,383	47%
2	$\theta_1 = 0.05$	39,290	7,948	3,417	42%
3	$\theta_1 = 0.02$	39,886	7,881	3,373	43%
4	$\theta_0 = 0.6$	7,146	2,132	1,064	50%
5	$\theta_0 = 0.6$, $\theta_1 = 0.05$	39,396	2,626	1,265	48%
6	$\alpha = 0.10$	9,581	16,525	6,437	39%
7	$\alpha = 0.10$, $\theta_1 = 0.05$	39,142	15,839	6,123	39%
8	$\alpha = 0.10$, $\theta_1 = 0.4$	4,015	8,947	3,922	44%
9	$\alpha = 0.10$, $I_3 = 6hr$, $I_4 = 6hr$	3,126	5,853	1,729	30%
10	$I_3 = 24hr$, $I_4 = 24hr$	1,350	57	11	19%

Table 5.3: Experimental results using TRW showing FP (false positives) and FD rate (false discovery rate[4]).

with high network traffic volume, impose a new challenge for network administrators: choosing appropriate values according to the network nature or environment. Another observation is the low number of remote hosts classified by TRW as either scanners or benign. In all the experiments, TRW classified less than 20% of remote hosts initiating connection attempts leaving over 80% unclassified, pending further events.

The first experiment uses the default TRW parameters as in the first row in Table 5.3 which shows that about half of detected scanners are false positives (relative to $TRWaGT_{FP}$). 2,383 IP addresses are marked by TRW as scanners but they were not in the set A and thus these remote hosts only attempted connections to the ports that were observed at one time to offer a service. The fact that some of their connections failed are due to temporarily unavailable services and not because they are scanning the network. Upon tracking down several local hosts of the network services causing failed connections, we found the typical scenario occurs after the local host gets disconnected from the network for some time and thus its corresponding services become inaccessible.

As discussed in Section 5.1, several possible factors can cause a local host to be disconnected from the network (e.g., restarted after a system update or turned off).

[4]Note that the reported false discovery rate (FD rate) in Table 5.3 is different than false positive rate (FP rate) as explained in Section 5.3.2.

If a remote host is not in the TRW benign list and happens to attempt connecting to any of the services in this temporarily unavailable local host, then it will get penalized by TRW for making failed connection and its likelihood ratio will be raised towards being classified as a scanner. In some other cases, the local host was reachable but some of its network services were not responding. The most likely scenarios are that either the application running the network service was closed or uninstalled. Consequently, when the remote host made unsuccessful connection attempts, such that its likelihood ratio exceeded the upper bound, it was classified as a scanner by TRW. In most cases, however, we found that those network services causing failed connections became active again as a result of either the users turning the corresponding hosts on, reconnecting to the network, or restarting the related applications.

Since θ_1 models the probability a scanner remote host with no prior knowledge of the targeted network initiates a successful connection, we reduced the TRW default value of θ_1 to 0.05 in the second experiment. As expected, this change slightly increased the number of reported scanners. However, the number of reported benign sources increased by about 4 times since now only two consecutive successful connections are required by TRW to classify a source as benign. 3,417 remotes in the TRW scanner list S are not in set A and thus the false positives represents 42% of the number of detected scanners. Similar results are obtained by changing θ_1 from 0.05 to 0.02 in the third experiment.

Our analysis of the network traffic shows the transient nature of the offered services which suggests that the probability of a benign remote host initiating a failed connection is actually high compared to the intended probability in TRW default settings. Therefore, to decrease the number of false positives, we decreased θ_0, the estimated probability a benign remote host initiates a successful connection, to 0.6 in the fourth experiment. However, although the number of scanners decreased to less than half that in the first experiment, about 50% of the detected scanners are false positives. Adjusting θ_1 and α in experiment 5, 6, 7, and 8 did not help either in decreasing the number of false positives.

In experiment 9 and 10, we increased the time window for keeping the status of successful and unsuccessful connections initiated by remote hosts. The FD rate in

experiment 9 decreased to about 30%. Similar configuration in experiment 10 also reduced the FD rate to 19%, however, the number of reported scanners diminished to 57 only. The reduction in the number of false positives is because of the increase in the expiry time (I_3 and I_4) for the entries in the sets FC and SC, increasing the duration used for keeping the state of connections for inbound traffic. Knowing that a remote host r has unsuccessful connection with a local host l in the past and keeping this information long enough prevents TRW from re-penalizing r for making another unsuccessful connection with the same host, l. This would have happened if the previous failed connection was not found in the set FC (as in line 11 in Algorithm 4) and Λ_r will be raised towards being classified as a scanner, which explains the slight reduction in the number of false positives in the experiments 9 and 10. On the other hand, knowing that r connected successfully to l in the past and keeping this information in the set SC will not prevent TRW from penalizing r if r happens to contact l while either the network service in l or the host itself is temporarily unavailable (as in line 14). Considering if the current unsuccessful connection is in SC first before increasing r's likelihood ratio (towards being classified as a scanner) will help in reducing the number of false positives.

Choosing an appropriate expiry time for the entries in the sets FC and SC is challenging since it depends on how often client applications connect to the network services. Keeping the entries for too long consumes considerable memory since the status of every inbound connection, whether successful or not, together with the source and destination addresses need to be stored. Also, choosing long values for I_1 and I_2 will not reflect the possible change of state in remotes from benign to scanner and vice versa since the algorithm does not change the likelihood ratio of the classified remote hosts (line 8 and 9).

STRW. The NS table in STRW differs from the one used in obtaining $TRWaGT_{FP}$ as defined in Section 5.3.2. Notice that while the table of network services used in $TRWaGT_{FP}$ includes all services offered at any time during the dataset capture period, an instance of NS contains only the active services during I_7 time window. For the dataset described in Section 5.3.1, we first set I_7 to three days (mainly to

take into account machines possibly turned off on weekends). STRW1 in Table 5.4 shows the results of repeating the same previous ten experiments on STRW with I_7 set to three days. Relative to $TRWaGT_{FP}$, the number of false positives decreased in all experiments compared to TRW by 35% on average. While this is a substantial improvement, FD rate is still high (an average of 34% of detected scanners by STRW are false positives). Although the number of scanners is also decreased by 25% on average, the reduction is more than the difference in the number of false positives for all the experiments. That is, K = (no. of scanners in TRW − no. of scanners in STRW) − (no. of FP in TRW − no. of FP in STRW) > 0. The difference, K (an average of 673 scanners per experiment), represents additional false positives in TRW (unconsidered in Tables 5.3 and 5.4) that can be added to our original discovered false positives (i.e., $TRW\ aGT^*_{FP} - TRWaGT_{FP}$, as in Section 5.3.2).

These remote hosts made failed connection attempts with both IP/port tuples both in the NS table and IP/port tuples not in the NS table (i.e., open and closed ports in the monitored network), but unlike TRW, the number of connection attempts to distinct IP/port tuples not in the NS table is not high enough to trigger STRW to report them as scanners. On the other hand, the number of remote hosts declared to be benign is more in STRW than TRW in every experiment by 3% on average. This represents remote hosts that made enough successful connection attempts to be flagged as benign but also made failed connection attempts to NS entries which made TRW report them as scanners or pending.

There are two reasons that STRW1 FD rate is still not low: (i) services temporarily unavailable for more than three days increase the number of false positives; and (ii) time is required for the NS table to get populated with available services. Hence, it is preferable to increase I_7 and to run STRW for a few days (i.e., for a period of I_7) prior to considering its scan detection results. Consequently, in STRW2, I_7 is set to one week and STRW is run three days before considering the scan detection data (March 12-14). After the first three days (by beginning of March 15th), the NS table was populated by 1097 network services and all other variables were set back to their default values. Table 5.4 shows the results of repeating previous experiments using STRW2. The average number of false positives in STRW2 decreased by 77%

Exp.	Non-default parameters - I_1 and I_2 are not used - $I_6 = 24hr$	no. of benign remotes		no. of remote scanners		no. of FP (distinct scanners)		FD rate (distinct scanners)	
		STRW1	STRW2	STRW1	STRW2	STRW1	STRW2	STRW1	STRW2
1	None	9,993	9,999	4,290	3,143	1,757	633	41%	20%
2	$\theta_1 = 0.05$	39,682	39,708	6,525	4,872	2,496	881	38%	18%
3	$\theta_1 = 0.02$	40,002	40,024	6,513	4,862	2,494	881	38%	18%
4	$\theta_0 = 0.6$	7,203	7,206	1,866	1,333	814	291	44%	22%
5	$\theta_0 = 0.6$, $\theta_1 = 0.05$	39,731	39,751	2,323	1,688	984	360	42%	21%
6	$\alpha = 0.10$	9,938	9,946	12,122	9,592	3,785	1,300	31%	14%
7	$\alpha = 0.10$, $\theta_1 = 0.05$	39,601	39,628	11,977	9,470	3,762	1,299	31%	14%
8	$\alpha = 0.10$, $\theta_1 = 0.4$	4,379	4,383	6,695	5,018	2,519	883	38%	18%
9	$\alpha = 0.10$, $I_3, I_4 = 6hr$	3,301	3,304	3,896	3,186	1,195	501	31%	16%
10	$I_3 = 24hr$, $I_4 = 24hr$	1,405	1,404	22	20	2	0	9%	0%

Table 5.4: STRW experimental results. FP denotes false positive and FD rate denotes false discovery rate[5].

compared to TRW in Table 5.3. The FD rate in STRW2 (an average of 16%) is also substantially less than TRW. In addition, the number of scanners decreased by an average 43% compared to TRW. On average, $K = 700$ remote hosts declared to be scanners per experiment (i.e., additional false positives in TRW to be added to FD rate in Table 5.3). As discussed in Section 5.2.3, the required memory for the NS table was insignificant (less than 3000 entries of IP/port tuples at any given time) as it is bounded by the number of offered network services.

5.4.2 Class B Dataset

In this section, we only report the main findings of repeating the same previous ten experiments using both the TRW and STRW algorithms on the class B dataset. This is due to the relative similarity between the class C dataset (a detailed discussion of this dataset results is given in Section 5.4.1) and the class B dataset (this section) in the comparison results between TRW and STRW performance in terms of detection accuracy.

With the default TRW settings, a remote host making consecutive failed attempts to 4 or more internal hosts in the target network will be reported as a scanner. Hence, according to $TRWaGT_{FP}$ (see Section 5.3.2), having less than 4 internal hosts offering

[5]Note that the reported false discovery rate (FD rate) in Table 5.4 is different than false positive rate (FP rate) as explained in Section 5.3.2.

network services implies that there will be no false positives using TRW default settings. However, benign remote hosts which make failed connection attempts to temporarily unavailable services in addition to some non-existing services (at least one but less than 4) might be mistakenly flagged as scanners by TRW (i.e., $TRW\ aGT^{*}_{FP}$; see Section 5.3.2) even if there are less than 4 offered services. Therefore, although the probability of having false positives in TRW increases with more network services, STRW is expected to perform better than TRW even with few offered services.

Considering $TRW aGT_{FP}$ only, the FD rate for TRW was less than 1% for all 10 TRW experiments corresponding to those in Section 5.4.1. In all the experiments, both STRW1 and STRW2 had no false positives. However, the number of scanners in STRW1 and STRW2 was less than TRW due to some failed connection attempts that were made to temporarily unavailable network services and thus STRW did not consider them in updating the likelihood ratio of the corresponding remotes towards being classified as scanners. For $TRW\ aGT^{*}_{FP}$, the highest FD rate for TRW is in experiment 8 (24%) in which it requires only two consecutive failed connection attempts to classify a remote host as a scanner. Using STRW2 in this second dataset, on average 94% of TRW false positives in all experiments are eliminated. All the scanners detected by STRW are also detected by TRW.

While the detection accuracy of STRW over TRW in this dataset is not as substantial as in the previous dataset, STRW remains a better choice in such (more static) environments for several reasons. First, sudden changes in the availability of the network services do not add false positives. Even main servers in enterprise environments may occasionally lose network connectivity due to various causes: software patches, hardware upgrades, power outage and network devices failures. Second, taking into account that network devices (e.g., routers and switches) and IDS detectors might skip packets that cannot be processed in real time in some congested subnets of the network, some connection attempts might erroneously be interpreted as unsuccessful (e.g., uncaptured SYN-ACK packets) which will cause false positives in TRW.

5.5 Concluding Remarks

Using failed connection attempts as an indication of network scanning activity remains a feature that scanners can not evade since their lack of knowledge of the accessible network hosts or services is what drives them to scan a network. The relatively subtle modifications to TRW introduced by STRW lead to dramatic reduction in false positives. In our empirical evaluation of TRW on network traces from two sites, across different configurations of parameters, between 19% and 50% of the scanners reported by TRW are false positives. While maintaining the same detection accuracy as TRW, STRW significantly reduces the false discovery rate (29% and 77% of TRW false positives are avoided in two datasets studied). We also provide guidelines on how to set appropriate values for the parameters of both TRW and STRW to enhance detection accuracy.

While using the EM technique as a proxy for approximated ground truth can only identify a subset of the scan detection false positive results, its high confidence level in identifying benign remote hosts makes it suitable as a preliminary evaluation method for scan detectors. We hope that by improving scan detection accuracy our work may encourage broader adoption of scan detection mechanisms and thus enable early defensive actions for mitigating network attacks that are preceded by such a reconnaissance phase.

Chapter 6

Evaluation in the Absence of Absolute Ground Truth: Towards Reliable Evaluation Methodology for Scan Detectors

Although network reconnaissance through scanning has been well-explored in the literature, new scan detection proposals with various detection features and capabilities continue to appear. To our knowledge, however, there is little discussion of reliable methodologies to evaluate network scanning detectors. In this chapter, we show that establishing *ground truth* labels of scanning activity on non-synthetic network traces is a more difficult problem relative to labeling conventional intrusions. The main problem stems from lack of absolute ground truth (AGT). We identify the specific types of errors this admits. For real-world network traffic, typically many events can be equally interpreted as legitimate or intrusions and therefore establishing AGT is infeasible since it depends on unknowable intent. We explore how an estimated ground truth based on a discrete classification criteria can be misleading since typical detection accuracy measures are strongly dependent on the chosen criteria. We also present a methodology for evaluating and comparing scan detection algorithms. The methodology classifies remote addresses based on continuous scores designed to provide a more accurate reference for evaluation. The challenge of conducting a reliable evaluation in the absence of AGT applies to other areas in network intrusion detection, and corresponding requirements and guidelines apply.

6.1 Introduction

Reliable evaluation of an anomaly detection system, while highly desirable, is typically challenging due to the difficulty in validating the detection results of the anomaly detector relative to a typical misuse-based detector [99]. Despite the numerous network

scan detection algorithms and tools in the literature, there is no known evaluation methodology that can be reliably used to assess the accuracy, capabilities, and limitations of a given detection algorithm. Simulation and emulation approaches can only be used to generate certain classes of scanning since general signatures for all network scanning events do not seem possible. Also, simulated network and Internet traffic may not realistically represent real-world traffic. An alternative, community-based, testing approach could be to adopt some appropriate datasets with the prospect that such publicly available datasets might slowly be labeled by different approaches and become a standard set of datasets for community testing and validation. However, traces of real-world network traffic that are publicly accessible are limited, including due to privacy and legal reasons, or out-dated. In fact, generally in network intrusion detection, two publicly available synthetic datasets [62, 61, 3] continue to attract the majority of attention, despite containing only a specific set of typical attacks and being known to be inappropriate [63, 65, 116].

A common evaluation approach in network scanning detection is to compare results with those obtained by a state-of-the-art algorithm for scan detection that the evaluator considers a fuzzy gold standard. However, not only is this reference standard implicitly deemed to be perfect, which removes the need for a new detection algorithm, but also better detection in the new algorithm relative to the reference algorithm cannot be easily identified, as it requires another reference to corroborate.

There are many different definitions of a "port scan", e.g., "*[a port scan] denoted* $C = (s, T, \triangle t)$*, is a set of connection attempts from source address* s *during time interval* $\triangle t$ *where* T *is the set of targets (address, port) and whose purpose is reconnaissance*" [34]. Extracting an accurate signature from such definitions is problematic from an intrusion detection perspective. The challenge is in capturing the *intent* of a connection attempt, to distinguish scanning from "normal" connection attempts. Consequently, this often precludes validating the classification of what a scan detection algorithm has reported as scan events.

In this chapter, we examine the differences between conventional intrusion detection systems and scan detectors. Given that it is possible to scan a particular network

with traffic that adheres to network protocol specifications both syntactically and semantically, we argue that no particular signature exists for individual scans. Based on individual scans, it is inevitable that any *ground truth* for a particular non-synthetic dataset will be imperfect. While it is possible to generate a perfect ground truth of a synthetic dataset, not only may the dataset traffic be an unrealistic representation of real-world network traffic, but also the generated scan events may represent only a subset of the existing patterns in real-world data. Therefore, for real-world network traffic, given that the computation of the ground truth of intrusions that might resemble legitimate traffic (e.g., see [64]) will inevitably include some errors, such uncertainties have to be considered in the evaluation process. We present an evaluation methodology for scan detection based on generating an aggregate classification score for each remote host. The behaviour of each host including both normal and abnormal activity over the time frame of a given network dataset is examined to generate the corresponding score. Our methodology is based on a numeric prediction of scanners intended to give a more accurate reference for evaluation under lack of AGT.

From studying existing real-time scan detection heuristics, we present a set of behavioural heuristics to demonstrate how to derive a ground truth reference (GTR) that assigns continuous scores (vs. discrete values) to remote addresses in a given real-world network trace. Such heuristics aim to determine normal and abnormal activity with respect to scanning and thus to calculate the aggregate anomaly scores for remote hosts. While our set is by no means comprehensive in terms of its coverage of all possible forms of scanning, its elements are chosen relative to a specific definition of scanners. Accordingly, the methodology is designed to be extendible by adding new behavioural heuristics to capture new classes of scanning. We show that different detection accuracy measures could contradict each other and that statistical metrics which measure the distance between the results of a given detector and those obtained by our imperfect ground truth (i.e., GTR) provide a more accurate evaluation.

While the detection accuracy is a crucial factor for evaluation and comparison, understanding the capabilities and limitations of a scan detector must be part of the evaluation. For example, the speed of detection in terms of the size of exposure

window in which a remote can scan the monitored network before being flagged as a scanner may play an important role in post-detection responses. Also, the susceptibility of the detector to evasion and attacks is an important factor for the practicality of deployment. As some prior literature has discussed such properties and attacks (e.g., see [105, 46, 43, 85]), we omit their description herein. Rather, we focus on how to evaluate detection accuracy using real-world traces in spite of the lack of AGT of scanning events. In addition to forming a GTR for evaluating and comparing network scanning detection mechanisms, the scanning activity identified by our methodology can be used to study the scanning behaviour or the distribution of malware over an extended period of time, or over large IP address spaces.

We summarize our main contributions as follows. To our knowledge, we are the first to explore the problems that can arise when evaluation is based on a GTR rather than AGT. We model the problem of performing evaluation in the absence of AGT, and analyze the requirements of using a GTR for either evaluating one intrusion detector or comparing multiple detectors. We identify the drawbacks of existing approaches for evaluation and comparing network scan detection mechanisms, and discuss challenges specific to scan detection evaluation that are absent in typical intrusion detection systems. We show that a ground truth (i.e., classifying remote hosts or connection attempts as either scan-related or benign) that is based on a discrete classification criteria could be misleading since the results of typical evaluation metrics will be dependent on the chosen criteria. We present a new evaluation approach for scan detectors designed to address uncertainties in GTR. We hope our evaluation methodology will be of use in evaluating and comparing existing scan detection mechanisms. While by no means fully addressing all questions that arise, we hope that our work will spur further exploration of the evaluation challenges involving a derived GTR. Ideally, this will lead to developing a better methodology for evaluating network security mechanisms in which AGT is unknown.

Organization. Network scanning background and motivation are given in Section 6.2. Section 6.3 analyzes the underlying requirements of using a GTR. Section 6.4 discusses the appropriateness of using some classical methods for obtaining a GTR in

scan detection. Section 6.5 examines the utility of typical intrusion detection discrete accuracy metrics in the absence of AGT of intrusions and present a distance-based continuous metric for evaluation with unknown AGT. To demonstrate establishing a GTR of continuous scores that can be used in the proposed accuracy metric, we present a new scan detection heuristic in Section 6.6. Section 6.7 concludes.

6.2 Background and Motivation

This section first discusses the importance of specifying which network scan definition and type(s) to use in the evaluation. It then discusses known mechanisms to obtain evaluation datasets, highlighting the key motivation for this work.

To build and evaluate detection heuristics for network scanning, a precise definition of a scanner is required. There are several, sometimes inconsistent, definitions of network or port scans. For example, Fyodor [33] defined scanning as "a method for discovering exploitable communication channels. The idea is to probe as many listeners as possible, and keep track of the ones that are receptive or useful to your particular need". De Vivo et al. [26] defined TCP port scanners as "specialized programs used to determine what TCP ports of a host have processes listening on them for possible connections". In the absence of a universal definition, in this chapter, we define network scanning as: the process of sending connection attempt packets to one or more ports at one or more hosts for the purpose of discovering if the host(s) is active, if the port(s) is open, or to gather information about the offered network service(s).

Under our definition of network scanning, various types of network scanning are identified in the literature. The most common type is *vertical scanning* which is probing a set of ports on the same IP address to identify the running network services on the IP address or to find out if the corresponding host is active. *Horizontal scanning* (also called *port sweeping*) is the process of probing multiple IP addresses for the same port to identify the running network services or to identify active hosts. The scanner may also probe multiple IP addresses for multiple ports (*strobe scan*) or probe multiple IP addresses for all ports (*block scan*). For evaluating a scan detector, it is important to specify which scanning types the detector is evaluated against (and

under which Internet protocols), including subtypes, e.g., a vertical scan detector might only target connection attempts starting with a SYN packet (also called *SYN-scan*). The evaluation ground truth data should include only the targeted types of scans.

Several heuristics were proposed in the literature for detecting known scanning behaviours rarely observed in benign traffic. Heuristics are often developed for real-time scan detectors and thus oriented towards fast detection with as few false positives as possible. The following are commonly used detection heuristics: (i) contacting non-existing local hosts or network services [98, 40]; (ii) making unsuccessful connection attempts since, unlike legitimate network traffic, most connection attempts that are part of reconnaissance activities are expected to fail given that active network services are unknown prior to scanning [49, 45, 36]; (iii) exchanging low volume of data with local hosts [36, 131]; (iv) contacting many local hosts or ports in a small time window [98, 40, 90, 91, 2]; and (v) contacting rarely accessed local hosts or ports [58, 105, 50].

The reminder of the section discusses evaluation datasets. Ideally, non-synthetic datasets labeled to identify the target classes of intrusions or anomalies might be used to evaluate the detector results. However, for scan detectors, not only are there no publicly available non-synthetic labeled datasets, but there are also no synthetic labeled datasets. In fact, the problem is not simply that currently no such labeled datasets are available, but rather that for scan detection, it is not clear that such labeling is reliable, given the impossibility of determining the intent of a connection attempt (see Section 6.4.1). The following subsections summarize known mechanisms used in the literature to obtain network traffic for evaluation.

Simulated Datasets. In simulation approaches, network traffic is generated by custom network simulation software to model real network behaviour of a particular configuration. Network simulation helps in verifying the correctness of a new or existing network intrusion detector and in predicting its performance, especially in complex network settings. Using simulation, both the network traffic and broader scanning campaigns are simulated. In addition to avoiding the legal and privacy

issues regarding using real-world traffic, simulation enables testing of a scan detector using various network configurations while controlling the characteristics of the scans.

On the other hand, simulation has known shortcomings. First, given that network traffic is diverse and variability is expected in both network-level (including background traffic) and application-level protocols [99], the simulated traffic may not realistically represent real-world traffic. It is important to test detectors on traces from operational networks that resemble those where the detectors will be deployed. Simulation must include Internet traffic that is hard to simulate or model due to heterogeneity, scale, and rapid change in the Internet's dynamics [30]. Second, the generated intrusions or reconnaissance activity (i.e., scan events) might only represent a subset of existing patterns in the wild. The inadequate understanding of many anomalies and reconnaissance activity make it hard to characterize them precisely [88].

Simulating network scanning activity is even more challenging because: (i) based on the remote host's intent, an inbound connection attempt can be considered either as a scanning activity or benign; and (ii) it is not necessary that network traffic originated from a scanner contains or is associated with malicious payloads.

Emulated Datasets. In network emulation, a subset of the studied traffic is real-world data. For example, starting with a real-world dataset, scanning events could be simulated to the target IP address range and then injected in the dataset. Emulation approaches seem better than simulation because network traces can be collected from an environment representative of where the detector will be deployed. However, in emulation, the starting dataset could include scanning data as well which will affect the evaluation of the detector; filtering out such scanning traffic requires an accurate scan detector in the first place, and therefore full knowledge of ground truth. Also, the feedback effects of the interaction between the simulated scanning events and the real-world network traffic are not captured. In addition, similar to simulation, the generated scanning traffic might not reflect the variety of scanning patterns in the wild. Both simulation and emulation approaches are known to be a good starting point to test the detector ability to identify certain types of intrusions but not sufficient in its own to evaluate the detector accuracy and capabilities [87].

Real-world Traffic Datasets. Given the drawbacks of simulation and emulation, and the lack of publicly available non-synthetic accurately labeled datasets, researchers usually evaluate detectors using their own network traffic. Such gathered datasets provide real-world data of both network traffic and intrusions (e.g., scanning events). While anonymization of these datasets (e.g., see [4]) helps substantially in removing private and other sensitive information, information leakage is still possible [25] and thus for ethical, privacy, and legal reasons, such datasets often stay private.

Such collected datasets are subject to the following issues: (i) the network environment of the dataset may exhibit artifacts that until identified and explained, significantly affect detector performance; (ii) the nature and size of the network might not represent a realistic target deployment environment; (iii) with the lack of control over the characteristics of the scan events, it may be hard to determine the conditions under which a detector will perform well; and (iv) obtaining a reliable ground-truth of the existing intrusions that the detector is designed to detect is challenging and might be infeasible for some types of intrusions. The next section explores the challenges in establishing a ground truth of intrusions in real-world traffic data and formulate the requirements under which a ground truth reference can be used. We use network scanning as an example.

6.3 Ground Truth Reference with Uncertainties: Formalizing Properties

The major obstacle in using real-world traffic data is obtaining a reliable ground-truth of the intrusions that the detector is expected to detect. For evaluating anomaly detection systems, an absolute accurate ground truth (AGT) of intrusions is an ideal reference in which all true positives are correctly identified. However, generating such AGT data is usually unattainable as it implies that the malicious intrusions/anomalies in question can be perfectly modeled either automatically or manually. In this section, we analyze the requirements of using an imperfect ground truth reference for either evaluating one detector or comparing two or more detectors. We also explore some methods for combining two or more ground truth references.

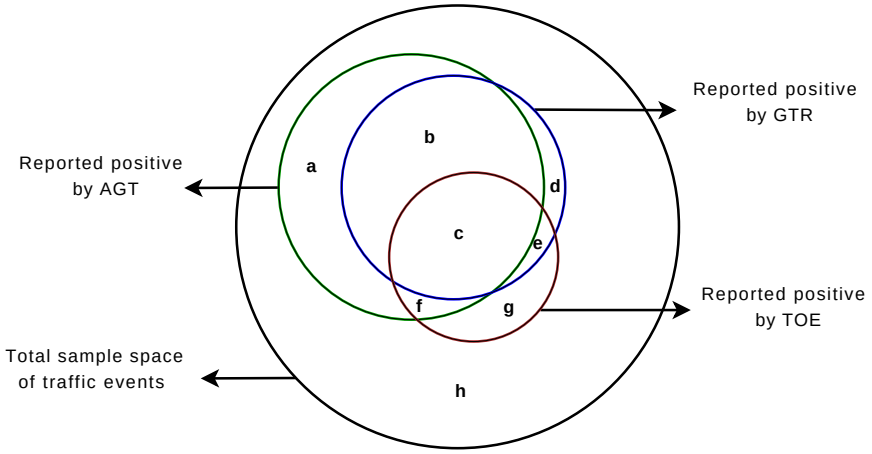


Figure 6.1: Representative Venn diagram showing possible intersections between AGT, GTR, and TOE.

	Base	True Positives	False Positives	True Negatives	False Negatives
AGT	AGT	a+b+c+f	∅	h+d+e+g	∅
GTR	AGT	b+c	d+e	h+g	a+f
TOE	AGT	c+f	e+g	h+d	a+b
TOE	GTR	e+c	g+f	h+a	b+d

Table 6.1: Sample classification by AGT, GTR, and TOE corresponding to Figure 6.1.

6.3.1 Ground Truth Reference for Evaluating One Detector

In the absence of a AGT, it is essential to establish a ground truth reference (GTR) that can provide a reliable reference baseline or an estimated ground truth of intrusions/anomalies. In this section, we pursue the following key questions. Q1: For a given GTR, what are the underlying requirements that the GTR must fulfill? Q2: Are there desirable GTR properties according to the evaluated detector?

Figure 6.1 shows the possible intersection between AGT, GTR, and a TOE (target of evaluation, i.e., the detector to be evaluated). True/false positives and true/false negatives are given for AGT, GTR, and TOE in Table 6.1. There are many cases where a given GTR is inappropriate to evaluate the TOE. We define two invalid cases. (Notation: AGT_p and AGT_n, respectively denote reported positive and negative by AGT; likewise for GTR and TOE):

i) Case 1: $(TOE_p \cap AGT_p \neq \emptyset) \wedge (TOE_p \cap GTR_p = \emptyset)$. In this case, all detected positives by TOE relative to AGT are considered false positives relative to GTR; i.e., $f \neq \emptyset$ and $(c + e) = \emptyset$.

ii) Case 2: $(TOE_p \cap AGT_p = \emptyset) \wedge (TOE_p \cap GTR_p \neq \emptyset)$. Here all detected positives by TOE are actually false positives relative to AGT but some are considered true positives if evaluated according to GTR; i.e., $(f + c) = \emptyset$ and $e \neq \emptyset$.

Ideally, the closer GTR is to AGT, the more accurate the evaluation, where the best case is $GTR = AGT$. Therefore, we want to maximize c and minimize f, e, d, and a. If $f > c$, GTR will underestimate both the *true positive rate* (*TPR* or *detection rate*) and the *false positive rate* (*FPR*) of TOE. If $e > c$, GTR will overestimate TPR and FPR of TOE. Therefore, it is desirable that both f and e are close to $\emptyset$. The larger the set a, the smaller FPR and the higher TPR from their correct values (i.e., their values relative to AGT). Also, it is desirable that, relative to AGT, the number of GTR true positives is more than the number of TOE true positives ($(b+c) > (f+c)$ and thus $b > f$). While this is a theoretical analysis, as AGT is unknown in practice, there are some cases where it is possible to obtain a AGT or a more accurate GTR but through a costly process (e.g., see Section 6.4.1). In such cases, the AGT might be obtained once, say on one dataset, to evaluate different mechanisms for generating GTRs. The more accurate GTR that meets the conditions above is then used for evaluating a TOE on other datasets. Also, if there is a mechanism to obtain a AGT that is costly and perhaps infeasible to perform over the whole sample space, it might be possible to use random sampling to estimate the intersection between GTR and AGT, and thus examine the requirements for using GTR to evaluate TOE.

6.3.2 Ground Truth Reference for Comparing Two or More Detectors

In typical intrusion detection evaluation, the performance and detection accuracy of a detector is compared to one or more other detectors. As discussed above, it is preferable that the GTR detection accuracy is significantly better than the evaluated detectors. That is, the coverage of the GTR's true positives and negatives must be sufficient to include most of those reported by TOEs either as positive or negative.

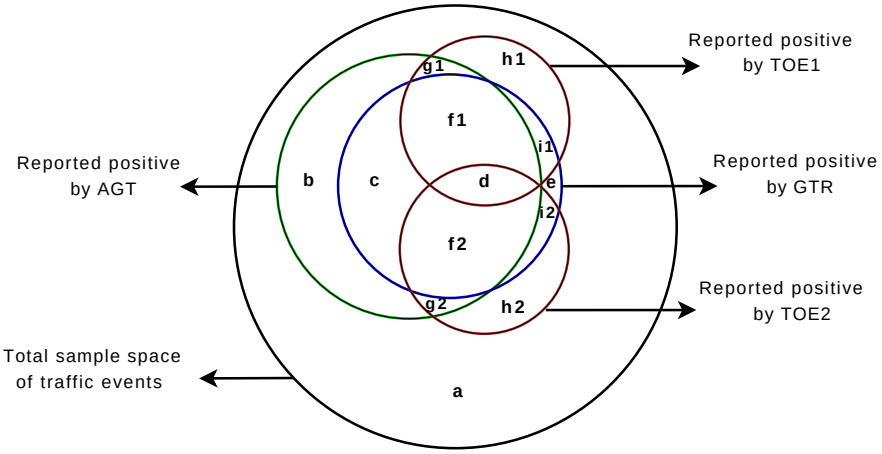


Figure 6.2: Representative Venn diagram showing possible intersections between AGT, GTR, TOE1, and TOE2.

	Base	True Positives	False Positives	True Negatives	False Negatives
AGT	AGT	$b+c+d+f_1+f_2+g_1+g_2$	$\emptyset$	$a+e+h_1+h_2+i_1+i_2$	$\emptyset$
GTR	AGT	$c+d+f_1+f_2$	$e+i_1+i_2$	$a+h_1+h_2$	$b+g_1+g_2$
GTR	GTR	$c+d+e+f_1+f_2+i_1+i_2$	$\emptyset$	$a+b+g_1+g_2+h_1+h_2$	$\emptyset$
TOE1	AGT	$d+f_1+g_1$	i_1+h_1	$a+e+i_2+h_2$	$b+c+f_2+g_2$
TOE1	GTR	$d+f_1+i_1$	g_1+h_1	$a+b+h_2+g_2$	$c+e+f_2+i_2$
TOE2	AGT	$d+f_2+g_2$	i_2+h_2	$a+e+i_1+h_1$	$b+c+f_1+g_1$
TOE2	GTR	$d+f_2+i_2$	g_2+h_2	$a+b+h_1+g_1$	$c+e+f_1+i_1$

Table 6.2: Sample classification by AGT, GTR, TOE1, and TOE2 corresponding to Figure 6.2.

Figure 6.2 shows possible evaluation errors when evaluating two detectors TOE1 and TOE2 using a AGT and a GTR. True/false positives and true/false negatives are given for AGT, GTR, TOE1 and TOE2 in Table 6.2. The invalid cases defined in Section 6.3.1 are applied to both TOE1 and TOE2. The desired GTR properties can be summarized in the following:

P1: Relative to AGT, the number of GTR true positives exceeds the number of true positives of each TOE. That is, $(c+d+f_1+f_2) > (d+f_1+g_1)$ and $(i_1+h_1) > (e+i_1+i_2)$. It follows that $g_1 < (c+f_2)$ and $h_1 > (e+i_2)$. Similarly, $(c+d+f_1+f_2) > (d+f_2+g_2)$ and $(i_2+h_2) > (e+i_1+i_2)$. It follows that $g_2 < (c+f_1)$ and $h_2 > (e+i_1)$.

P2: $(f_1 + d) > g_1$ and $(f_2 + d) > g_2$, since otherwise GTR will underestimate TPR and FPR of TOE1/TOE2 (see Section 6.3.1).

P3: $(f_1 + d) > i_1$ and $(f_2 + d) > i_2$, since otherwise GTR will overestimate TPR and FPR of TOE1/TOE2 (see Section 6.3.1).

P4: The difference between (g_1/f_1) and (g_2/f_2), and the difference between (i_1/f_1) and (i_2/f_2) should be minimized to reduce unfair comparison of detection rate between TOE1 and TOE2.

If any of these properties do not hold, using GTR for evaluating two or more detectors could lead to incorrect conclusions about the detectors' accuracy. For example, for P4, if TPR of TOE1 is better than TPR of TOE2 based on AGT, using GTR for evaluation might yield the opposite result if $g_1 > f_1$ and $g_2 < f_2$. As discussed in Section 6.3.1, while the assurance of such properties is often difficult because of the lack of AGT, obtaining AGT for a dataset of limited size or using data sampling might be possible.

6.3.3 Using More Than One Ground Truth Reference

If more than one GTR is available for a given intrusion detection problem, say $GTR_1, GTR_2 ... GTR_n$, we propose using statistical methods from machine learning to deal with samples of multiple labels; i.e., a classification problem where each input sample has a set of mutually exclusive labels, one of which is correct (e.g., see [42]). The objective is to obtain a single GTR that combines the results of $GTR_1, GTR_2 ... GTR_n$. While studying such mechanisms is beyond the scope of this thesis, we briefly mention two commonly used approaches.

Hard Labeling. By using a hard labeling method, the samples are assigned classes (e.g., positive or negative) based on GTRs classifications. *Majority voting* is one of the simpler and effective approaches [56] where an event j is considered positive if at least z (out of n) GTRs flag the event as positive; i.e., if 1 represents a positive event, and 0 a negative:

$$GTR(j) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} (GTR_i(j)) \geq z \\ 0 & \text{otherwise} \end{cases}$$

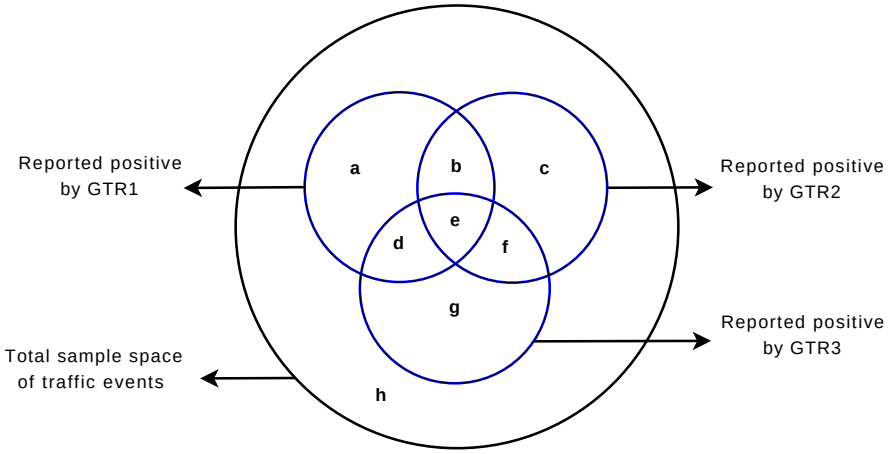


Figure 6.3: Representative Venn diagram showing possible intersections between three GTRs.

For example, if three GTRs are used to label the datasets as in Figure 6.3, the positives in the ground truth will be those in set e for $z = 3$ and those in sets b, d, f, and e for $z = 2$. z can be simply set as $\lfloor n/2 + 1 \rfloor$.

Soft Labeling. In soft labeling approaches, the excessive rounding in hard labeling is avoided by assigning input samples weighting factors that indicate their probability of belonging to a particular class. A weight of 1 indicates that all GTRs classified the sample as positive while a weight of 0 indicates that all GTRs classified the sample as negative. There are several advanced soft labeling methods for dealing with multiple labels in machine learning, e.g., using association rules to deal with multiple labels [111] (see also [42, 96]).

6.4 Obtaining a GTR

Evaluating intrusion detectors on real-world network traces requires establishing a ground truth of intrusions in the first place. This section explores the appropriateness of using four different approaches for obtaining a GTR of intrusions/ anomalies that can be equally interpreted as legitimate or malicious, the last of which is a variation that we build on. We use network scanning events as an example.

6.4.1 Manual Labeling

Manually labeling of traffic anomalies is a time consuming and costly process, as each connection attempt has to be inspected and labeled as either benign or malicious. Some legitimate network traffic might appear similar to scanning activity which makes it improper to deem a remote host a scanner from inspecting an individual connection even if the connection seems to be a scanning activity. Therefore, manual labeling requires examining as many as possible connection attempts of each remote to infer scanning intent which makes manual labeling of scanning events infeasible, especially for large datasets. Also, the labels assigned to the data by expert analysts cannot be confirmed to be correct.

If expert analysts are available, we recommend using some strict detection heuristics first to help identify possibly malicious connection attempts. These connection attempts can then be manually inspected and characterized by network expert analysts. While these heuristics must identify the majority of scanning events in a given dataset, the number of identified scanning events must not be too large to be manually inspected. If the number of identified scanning events by such initial heuristics are too large for manual checking, only the evaluated detector false negatives and positives (relative to the heuristics' results) can then be manually examined.

As discussed in Section 6.3.1, to examine whether a GTR is indeed appropriate to validate the TOE, random sampling is first employed to choose a subset of the detection results of both the TOE and some other detection mechanisms that serve as a GTR. Preferably, more samples are selected from the TOE false positives and negatives, relative to GTR.

Manual labeling is then used to verify this subset of samples. While the size of this subset depends on the available analysts' resources, a small subset might not be sufficient to give assurance of GTR appropriateness. Manual labeling can also be used for remote hosts that have a few number of connection attempts that exhibit both scanning and legitimate behaviour and thus the GTR may classify them with low confidence.

6.4.2 Using Signatures for Malware Known to Perform Network Scanning

Another possible way to label scanning events is to use IDSs to classify remote hosts with network traffic containing signatures for malware known to perform network scanning (e.g., worms and bots) as scanners (e.g., see [45]). While this technique can be used to capture a portion of scanners that exhibit such behaviour, it cannot be used as the sole mechanism to identify scanning events since scanners sending malicious payloads might represent only an insignificant portion of the overall number of scanners targeting a particular network. On the other hand, remote hosts with malicious traffic might not perform any scanning activity.

For TCP, a scanner is expected to send a malicious payload only when it finds a responsive service of interest to the scanner. Given that it is likely that the vast majority of the scanner's connection attempts fail, only the handshaking SYN packets will be captured from the scanner inbound traffic. Therefore, the presence of malware known to perform network scanning may only be used to confirm the detected scanners by other scan detection mechanisms.

6.4.3 Comparing With Another Scan Detector

A common method to evaluate a proposed scan detection mechanism in the literature is to compare its results with what is considered to be a state-of-the-art mechanism. Using another scan detector as a GTR can be beneficial if it uses different detection features than those used in the evaluated detector; however, this approach has the following drawbacks: (i) the detection of the state-of-the-art algorithm is deemed to be perfect and there seems to be no error-free scan detection technique; (ii) common shortcomings between the two algorithms might not be noticed; and (iii) superior detection results in the evaluated algorithm, relative to the other algorithm, cannot be easily identified. Therefore, this technique cannot achieve an independent evaluation on its own.

Alternatively, combining the detection results of multiple scan detection algorithms into one GTR might yield a reliable reference to compare against (see Section 6.3.3). The scan detection algorithms must be chosen carefully to complement the shortcomings of each other, preferably with different detection heuristics.

6.4.4 Long-term Behaviour Monitoring of Remotes

In establishing a GTR of scanners, we suggest making better use of the opportunity to exploit observation of the availability of the traffic of remote hosts contacting the target network over a relatively long period of time. That is, unlike real-time network scan detectors that require fast detection and thus a short monitoring window, a GTR (to be used primarily for evaluation, rather than real-time classification) can be built offline upon monitoring remote hosts over some appropriately long period of time (e.g., weeks or days rather than hours) and also over a large IP address space (if available). Note that the dataset used to evaluate a detector could be a subset of the one used to establish the GTR.

For clients behind a NAT (and also for some DHCP configurations), a single IP address could represent a set of actual hosts (this is also the case with proxies). This also means that the same remote host could use multiple IP addresses. In such cases, monitoring network traffic of one remote IP address over a relatively long period of time will not be effective since the IP address may represent multiple actual remote hosts. However, the effect of such limitation is proportional to the probability that more than one remote host contacts the network in question using the same IP address. A scanner behind a NAT might be missed due to using more than one IP address (i.e., a false negative case). Conversely, a detected scanner behind a NAT means that other benign remotes using the same IP address will also be considered scanners (i.e., a false positive case). However, most NATs consist of only a few hosts and it is usually possible to identify a remote that is behind a NAT (to treat its network traffic differently) [19]. Also, DHCPs often rotate IP addresses on the order of several days [19]. Moreover, if the evaluated detector relies on IP addresses for remote hosts identification and thus is not designed to consider such cases, then it seems fair to evaluate it relative to a GTR that will also miss these cases.

We propose adapting some of the existing and known effective scan detection mechanisms such that the network traffic of each remote address is examined for the entire available network traces before classification. As we discuss further in Section 6.6, given the possible similarity between legitimate and scanning behaviour in a given inbound connection attempt, the more traffic examined of the same remote

host that initiated the connection, the more can be inferred about the remote's intent. The intuition is that the repetition of what might be considered "abnormal" network traffic and the absence of "normal" traffic gives a signal of malicious intent. Therefore, for a given dataset, long-term monitoring of remotes' behaviour seems a promising strategy for obtaining a reliable GTR in the absence of AGT.

6.5 Comparing TOE Results with a GTR

Once a GTR is obtained, for example by one of the approaches of Section 6.4, the results from a detector being evaluated, TOE (i.e., target of evaluation), are compared with the GTR to evaluate the TOE performance. When AGT is unknown, considerable caution is required in comparing TOE results with those obtained by the GTR in terms of interpreting the disparity (see Section 6.3.1). Nevertheless, in numerous studies in the literature of intrusions/anomalies that resemble legitimate traffic (e.g., [45, 97]), a proposed GTR of a real-world dataset, while not conceived as perfect, is typically used as such implicitly in validating the accuracy of a TOE. In Sections 6.5.1 and 6.5.2, we study (to our knowledge for the first time) the suitability of typical detection accuracy metrics when AGT is unknown. In Section 6.5.3 we present a new score-based distance metric designed to provide a more accurate evaluation of detection accuracy. We use network scanning as a concrete case study.

6.5.1 Using Typical Detection Accuracy Metrics in Binary Classification

To date, an implicit assumption has been that a (perfect) ground truth is available for using typical detection accuracy metrics. In this section, we explore using the typical detection accuracy metrics in evaluating scan detectors, particularly when AGT is unknown. An intrusion detector is often a binary classifier (i.e., based on a binary granularity) in which samples are classified into two groups: (1) positive, representing an intrusion or attack; and (2) negative. Typically, when evaluating an anomaly detector, two critical questions are: (i) what instances did the detector mistakenly report positive? and (ii) what instances did the detector mistakenly report negative?

Basic accuracy measures and its functions. The false positive rate is the proportion that the instances in (i) represent of the total actual negative instances ($\mathbf{FPR} = FP/(FP + TN)$; TN denotes number of true negatives). Similarly, the true positive rate (also called *detection rate*, *effectiveness*, and *recall*) is the proportion that the instances in (ii) represent of the total actual positive instances ($\mathbf{TPR} = TP/(TP + FN)$).

As in most IDSs, there is a trade-off between FPR and TPR in network scanning detection: while making the scan detector more sensitive increases the chance of introducing more false positives, lowering false positives increases the chance of lowering detection rate.

FPR remains the most critical performance measure in intrusion/anomaly detection because of the high cost of collateral damage associated with penalizing legitimate traffic. In network scan detection, the FPR will be small if the number of true negative samples is significantly larger than the number of true positive samples. That is, regardless of the proportion of mistakenly reported scanners by the detector, the FPR is expected to be small. Given that the purpose of network scanning is often reconnaissance rather than imposing a direct threat such as carrying a malicious payload, the damage associated with false negatives seems less than that of false positives.

Other detection accuracy ratios include the *positive predictive value* ($\mathbf{PPV} = TP/(TP+FP)$; also called *efficiency* and *precision*), the *false omission rate* ($\mathbf{FOR} = FN/(FN + TN)$), and the *true negative rate* ($\mathbf{TNR} = TN/(TN + FP)$; also called *specificity*). The PPV is the proportion that the correctly reported positive instances represent of the total reported positive. FOR is the proportion that the mistakenly reported negative instances represent of the total reported negative. TNR is the proportion that the correctly reported negative instances represent of the total actual negative instances.

While TPR and FPR are essential metrics, from the above observations, FPR (and also TNR) is not expected to be useful for evaluating or comparing scan detectors because of the high number of true negatives. Alternatively, PPV gives a better measure of the efficiency of the detector in terms of false positives. Also, FOR gives a better measure of the number of true negatives versus the number of false negatives.

F-measure. A measure of accuracy that combines all four basic measures (TP, FP, FN, and TN) is the *F-measure* [115], which is a harmonic mean of TPR and TNR:

$$F - measure = \frac{2 \times TPR \times TNR}{TPR + TNR} \tag{6.1}$$

F-measure best value is 1 (when $TPR = TNR = 1$; i.e., when $FN = FP = 0$) and its worst value is 0 (when $TPR = 0$ or $TNR = 0$; i.e., when $TP = 0$ or $TN = 0$). While the F-measure represents a good accuracy metric, it is important to note that the F-measure is not sufficient in itself since a high or low F-measure value cannot be linked to either TPR or TNR (e.g., detectors A and B could have the same F-measure while having different TPR or TNR values). In scan detection, the TPR is expected to dominate Equation 6.1 since TNR is expected to be close to 1, and thus measuring TPR and TNR explicitly seems more useful than F-measure for evaluating scan detection accuracy.

Considering base-rate fallacy. Axelsson [13] pointed out that the base-rate fallacy (i.e., the basic rate of incidence) which may affect the FPR, is often not taken into account when evaluating an IDS. The emphasis is that the FPR should be calculated with respect to how many intrusions are expected in practice and not with respect to the maximum number of possible false positives. Axelsson introduced the *Bayesian detection rate* $P(T|D)$ (T denotes a true intrusion and D denotes a detected intrusion by the IDS) which is the probability that a detected intrusion is actually a true intrusion:

$$P(T|D) = \frac{P(T) \cdot P(D|T)}{P(T) \cdot P(D|T) + P(\neg T) \cdot P(D|\neg T)} \tag{6.2}$$

While the closer $P(T|D)$ is to 1, the higher the chance that all the detected intrusions are true positives, the closer $P(T|D)$ is to 0, the higher the chance that all the detected intrusions are false positives. Therefore, from Equation 6.2, the higher the probability of intrusions in practice $P(T)$, the lower the probability of false positives. Similarly, the higher the probability of no intrusions $P(\neg T)$, the higher the probability of false positives.

Much of the work in scan detection (e.g., see [45, 6, 97]) indicates that in many of the studied datasets, a significant part of the inbound connection attempts are destined to non-existing hosts or network services suggesting high scanning activity. However, the scanning activity is usually initiated by a small number of remote hosts relative to the total number of remotes contacting the monitored network. Given that the detection accuracy measures are based on the number of distinct remotes, from Equation 6.2, a low probability of a remote being a scanner means that there is a high probability of false positives in detected scanners. Thus, the actual FPR of a scan detector is expected to be higher than the reported one. Nevertheless, the probability of a remote being a scanner $P(T)$ depends on multiple factors, some of which are hard to quantify; for example, the number of network services' vulnerabilities at a given time and the popularity of the network services. Approximating $P(T)$ requires a large-scale study on several sites.

Summary. For measuring the detection accuracy for evaluation or comparison purposes, the set of accuracy metrics used in the evaluation should include all four basic measures (i.e., TP, FP, FN, and TN), as the detector accuracy cannot be fully quantified with only a subset of these measures. The appropriateness of an accuracy metric (i.e., a function of some or all four basic measures) depends heavily on the characteristics of the intrusion in question. Although it is hard to accurately measure the base-rate fallacy [13] for a particular intrusion/anomaly since it is hard to measure the probability of an intrusion/anomaly $P(T)$ occurring in practice, the base-rate fallacy [13] provides useful insights on the accuracy of other metrics (e.g., FP rate as discussed above).

Using any of the four basic measures or any of their functions for comparing TOE results with a GTR is under the assumption that AGT or a very accurate GTR exists for the studied dataset. This is because a binary GTR (i.e., based on a binary granularity; see Section 6.5.1) is sensitive to the chosen classification thresholds which could potentially result in misleading accuracy metrics. For example, for network scanning detection, if a binary GTR is based on the remote's percentage of failed connection attempts F so that the remote is considered a scanner if F exceeds a threshold z (e.g., $z = 50\%$), then the effect of each remote host with $F > z$ on the

detection accuracy metrics is equal. For two remotes a and b with F say equal to 60% and 100%, respectively, a and b are both equally considered positives in the GTR, although their likelihoods of being a scanner differ. Therefore, detecting (or missing) either a or b has the same effect on the detection accuracy metrics of a TOE. Consequently, we stress that typical detection accuracy metrics in binary classification are of limited value for evaluating or comparing intrusion/anomaly detectors using real-world network datasets in which AGT is unknown.

6.5.2 Comparison with Multiple Classes

Instead of building a binary GTR, remote hosts can be classified among three or more classes according to their observed level of suspicious behaviour. That is, remote hosts classified as scanners can be further divided into subcategories. For example, Simon et al. [97] assigned each {source IP, destination port} tuple to one of the following classes (a) *scanning*: if the network traffic sent to this port from the remote is considered scanning; (b) *normal*: for traditional client/server applications; (c) *P2P*: for peer-to-peer traffic; and (d) *noise*: for internet noise (e.g., backscatter). Also, to build a GTR for evaluating TRW, EM, and LQS, we used a classification criteria of six rules: *benign*, *likely benign*, *scanner*, *likely scanner*, *unknown-one* for those contacted unsuccessfully by only one host, and *unknown* for the rest (see Section 3.3.2 for further details regarding how the GTR is established and used for evaluation).

While this approach may provide a better classification criteria than a binary classification, it has the following shortcomings: (i) the granularity may not be sufficient since subclasses can be further derived; (ii) comparing the results of a binary classifier (i.e., the detector to be evaluated) with this multiclass criteria requires merging the results of some classes and/or eliminating others; and (iii) assigning remote hosts to definite classes that are used for evaluation requires an accurate GTR that might not be available for real-world network traffic.

6.5.3 Using a Standard Error Measure as a Distance Metric

In the absence of AGT, rather than performing a binary classification of remote hosts as either benign or scanners, or grouping remotes into a set of classes, we

propose assigning a score to each remote after analyzing its traffic over the entire dataset capture period. A maximum score of one is given for remote hosts with only abnormal traffic relative to a set of scan detection heuristics (see Section 6.6), and which therefore have a greater probability of being scanners. Likewise, a minimum score of zero is given for remote hosts with only normal traffic, which supports the belief that they are benign with a high probability. Remotes with a mix of normal and abnormal network traffic are given scores between zero and one accordingly.

For each TOE D (i.e., the detector to be evaluated), each remote host R is given a score (D_R) of one or zero according to the TOE classification (i.e., either a scanner or benign):

$$D_R = \begin{cases} 1 & \text{if the remote host is classified as a scanner} \\ 0 & \text{if the remote host is classified as a non-scanner} \end{cases}$$

RMSE vs. MSE. To compute the distance between the TOE and the GTR (i.e., the amount by which the TOE results differ from the GTR), we propose using the square root of the standard mean square error (RMSE) where the mean square error can be computed as follows: $MSE = \frac{1}{n}\sum_{i=1}^{n}(GTR_i - D_i)^2$; n is the total number of remote hosts in contact with the target network. RMSE gives an error measure that has similar units of measurement to those in the TOE scores, rather that the square of the units of the TOE scores as with MSE. The RMSE value ranges between zero and one. Ideally, an RMSE of zero means that the TOE has a perfect accuracy with respect to the GTR; this is unlikely in practice. Therefore, the closer the RMSE to 0, the smaller the difference between the GTR and the TOE and vice versa. In evaluating more than one TOE, the one with the lowest RMSE among all TOEs is the most similar to the GTR and thus its detection accuracy is the best among other TOEs.

Effect of outliers. Since the difference (or error) in the score of each IP address between GTR and TOE is squared, RMSE magnifies the effect of outliers. For example, if the score differences between GTR and TOE for two remotes are a and b, the effect of these two remotes to RMSE (i.e., adding $a^2 + b^2$ to RMSE) is smaller

than the effect of one remote with a difference of $a + b$ (i.e., adding $(a + b)^2$). In comparison with other error measures that treat all sizes of errors similarly (e.g., mean absolute error $= \frac{1}{n}\sum_{i=1}^{n} |GTR_i - D_i|$), RMSE gives us a better measure for the distance between GTR and TOE for the following reasons: (i) given that the greater the distance between the GTR and TOE scores, the more likely TOE has misclassified the remote host, whereas the smaller the distance, the more likely TOE has correctly classified the remote, heavily weighting outliers as in RMSE is desirable; and (ii) the GTR scores are already normalized between 0 and 1 which yields a consistent RMSE measure also between 0 and 1.

Detection rate versus the false positive rate with RMSE. Two question arise: (1) how can we use a distance measure like RMSE to evaluate or compare detectors? and (2) how can we evaluate the detection rate versus the false positive rate of a detector? The closer the RMSE value to 0, the better the performance of TOE relative to the used GTR. Similarly, for comparing two or more detectors, the one with the smallest RMSE measure is the one most close to GTR and thus it outperforms the other detectors. However, RMSE measure does not convey whether this is because the detector with the lowest RMSE value has a better detection rate, low false positive rate, or both.

In addition to proposing the use of the standard RMSE as an accuracy metric in the absence of AGT, we also suggest the decomposition of RMSE to allow visibility of the four contributing terms (TP, FP, FN, and TN), as they have different implications at different scenarios. Taking the GTR median score a as an arbitrary threshold for classification, the four basic measures can be computed as follows: (i) TP: TOE score $= 1$ and GTR score $\in (a, 1]$; (ii) FP: TOE score $= 1$ and GTR score $\in [0, a]$; (iii) FN: TOE score $= 0$ and GTR score $\in (a, 1]$; (iv) and TN: TOE score $= 0$ and GTR score $\in [0, a]$. In addition to the count of samples in each of these four categories, computing RMSE for the remotes (in contact with the monitored network) in each category individually (with n being the total number of remotes) gives the magnitude of error in each category ($RMSE^{TP}$, $RMSE^{FP}$, $RMSE^{FN}$, and $RMSE^{TN}$ denote the total RMSE for categories (i), (ii), (iii), and (iv), respectively). For instance, for a TOE, RMSE for a remote $R1$ in category (ii) with $GTR_{R1} = 0.1$ is greater than

RMSE for two remotes $R2$ and $R3$ in category (ii) with $GTR_{R2} = GTR_{R3} = 0.4$ since $(1-0.1)^2 > 2(1-0.4)^2$, and thus $R1$ increases $RMSE^{FP}_{TOE}$ (i.e., category (ii)) more than $R2$ and $R3$ combined.

As another example, for comparing two detectors, TOE1 and TOE2, Table 6.3 shows the differences between typical detection accuracy metrics and distance-based continuous metrics. Note that although TOE1 and TOE2 have the same TP rate (0.33), the fact that $RMSE^{TP}_{TOE1} > RMSE^{TP}_{TOE2}$ indicates that TOE2 has a better detection rate than TOE1. Also, note that although that TOE1 has a higher FP rate than TOE2, $RMSE^{FP}_{TOE1} < RMSE^{FP}_{TOE2}$. As discussed above, given that AGT is unknown, distance-based detection accuracy metrics give more accurate evaluation than typical detection accuracy metrics that depend on binary classification (see [10] for further discussion).

Summary. Using real-world network datasets for evaluation in which AGT is unknown, a GTR of intrusions/anomalies is typically established by giving each sample (e.g., a connection attempt or a remote IP address) a score(s) based on some detection heuristics. The sample is considered positive if its score(s) exceeds some threshold(s). Given that typical detection accuracy measures are functions from the four basic measures (TP, FP, FN, and TN) that are counted relative to a given GTR, regardless of the confidence levels of the GTR classification results, errors in the GTR classification have a large effect on the evaluation or comparison of TOEs. For a given GTR, the distance between the sample's score and the detection threshold is an important measure that reflects the confidence level in the GTR's classification correctness. The RMSE measures we propose in this section (i.e., $RMSE^{TP}$, $RMSE^{FP}$, $RMSE^{FN}$, and $RMSE^{TN}$) capture this distance so that rather than treating true/false positives/negatives equally, they are weighted according to the confidence level in their ground truth values.

6.6 An Example Heuristic for Establishing a Continuous GTR

In establishing a GTR of scanners for a given network dataset, time and computational resource constraints are less than those for a real-time detector. As discussed in Section 6.4.4, remote hosts' traffic over the entire dataset capture period can be

		GTR	$TOE1$	$TOE2$
Remote Hosts	R_1	0.6	1	0
	R_2	0.4	1	0
	R_3	0.9	0	1
	R_4	0.7	0	0
	R_5	0.4	1	0
	R_6	0.1	0	1
Typical Accuracy Metrics	no. of TP	NA	1	1
	no. of FP	NA	2	1
	no. of FN	NA	2	2
	no. of TN	NA	1	2
	TP *rate*	NA	0.33	0.33
	FP *rate*	NA	0.67	0.33
	FN *rate*	NA	0.67	0.67
	TN *rate*	NA	0.33	0.67
Distance Metrics	$RMSE^{TP}$	NA	0.16	0.01
	$RMSE^{FP}$	NA	0.72	0.81
	$RMSE^{FN}$	NA	1.3	0.85
	$RMSE^{TN}$	NA	0.01	0.32
	$RMSE$ (total)	NA	2.19	1.99

Table 6.3: An example of comparing two detectors with typical and distance-based accuracy metrics ($a = 0.5$).

examined to establish a GTR of scanners. While the detection features used to obtain a GTR should differ from those used in the evaluated scan detection algorithm, to avoid a circular argument, we argue that monitoring network traffic over a relatively long period of time and/or over a large IP address space breaks such circularity. That is, we argue that time duration is a distinct dimension in detection. Nevertheless, it remains desirable that the GTR includes detection features that are not used in the evaluated online detector. The requirements of using a GTR for evaluating a detector or comparing two or more detectors are discussed in Section 6.3. In this section, we present one detection heuristic as an illustration example of how to build a continuous GTR of remote scanners that assigns each remote host a score according to its observed network traffic, as described in Section 6.5.3.

$\boldsymbol{S_R}$:	the number of distinct {local_IP_addr, dst_port} tuples that R initiated successful connections to.
$\boldsymbol{F_R}$:	the number of distinct {local_IP_addr, dst_port} tuples that R initiated failed connection attempts to.
$\boldsymbol{O_R}$:	the number of distinct {local_IP_addr, dst_port} tuples that initiated successful or failed connection attempts to R.
$\boldsymbol{W_R^j}$:	the weight for the jth inbound successful connection that R initiated ($W_R^j = 1 - (1/n_i)$, where n_i is the number of local hosts with the port i open).
$\boldsymbol{X_R}$:	number of inbound connection attempts initiated by R that contain less than k outbound packets with only ACK flag set.

Table 6.4: Notation for Equations 6.3 and 6.4

Given that some scanning events might look like rare normal traffic if analyzed in isolation, repeated occurrences of what might individually be considered abnormal and the absence of normal traffic provide more confidence of malicious intent. Among the commonly used scan detection heuristics discussed in Section 6.2, to obtain a GTR of scanners, we employ two heuristics of abnormal traffic that seem hard to evade by scanners: (i) failed connection attempts initiated by the remote; and (ii) lack of data exchange, particularly outbound traffic to the remote. In addition, we employ two heuristics as a sign for normal traffic (iii) successful connections initiated by the remote; and (iv) connection attempts initiated by local hosts to the remote (whether successful or unsuccessful). While we combine these four heuristics in one main heuristic that captures many known scanning patterns, it is expandable by accommodating the addition of further detection heuristics.

For any connection attempt {remote IP address, local IP address, destination port} in a dataset, including both inbound and outbound traffic, only the first event involving this tuple is taken into account. Let n_i be the number of local hosts with port i open, the probability of a scanner to make a successful connection to the port i is $1/n_i$ (assuming random scanning). Therefore, a successful connection to port i is assigned a weight of $1 - (1/n_i)$, whereas a failed connection attempt to port i is always assigned a weight of 1. We combine the connection state heuristics (i.e., (i),

(iii), and (iv)) in one ratio $GTR1$ that is calculated for each remote R as follows (see Table 6.4 for notation):

$$GTR1_R = \frac{F_R}{F_R + \sum_{j=1}^{S_R} W_R^j + O_R} \tag{6.3}$$

The closer the $GTR1_R$ score value is to one, the higher the probability that R is a scanner. Similarly, the closer the score value of R is to zero, the higher the probability that R is benign. That is, failed inbound connection attempts increase $GTR1_R$ towards being a scanner, while successful inbound connection attempts and outbound connection attempts (whether successful or unsuccessful) decrease $GTR1_R$ towards being benign.

Unlike previous approaches, for the data exchange feature (i.e., (ii)), we use the count of successful inbound connections that contain at least k outbound packets with only ACK flag set (we call this count X_R). For a given remote, this is an indication of traffic sent by local hosts to the remote. Thus, the higher this count, the more data packets is sent to the remote. The higher the rate of inbound connections with low data exchange to all successful inbound connections initiated by the remote, the more evidence of malicious intent. Since the majority of successful TCP connections have many outbound packets with only the ACK flag set, a small threshold, say $k < 5$, indicates a low data exchange. Note that for failed inbound connection attempts, there will be no outbound packets with only the ACK flag set. We argue that this feature is particularly valuable to identify fortuitous scanners that have probed some active services since the data exchange with these services is expected to be minimal. The data exchange feature (GTR_2) is calculated as follows:

$$GTR2_R = \frac{X_R}{S_R} \tag{6.4}$$

The closer the $GTR2_R$ score is to one, the higher the probability that R is a scanner since this indicates a low data exchange in most connection attempts initiated by R. Similarly, the closer the $GTR2_R$ to zero, the higher probability that R is benign. The total score of R is derived by taking the maximum value of the scores obtained from

the detection features used in the evaluation:

$$GTR_R = max(GTR1_R, GTR2_R) \tag{6.5}$$

To evaluate a TOE D, for each R in an evaluation dataset, GTR_R is compared with D_R as explained in Section 6.5.3. While we argue that computing $GTR1_R$ and $GTR2_R$ over a relatively long period of time and/or over a large IP address space is sufficient to identify the majority of scanners, further detection features can be added to Equation 6.5 in a similar way, according to the TOE(s) in question.

6.7 Concluding Remarks

In this chapter, we discuss challenges in evaluating the accuracy of scan detectors and identify drawbacks of existing evaluation approaches. We present an evaluation approach for scan detectors. Driven by the similarity between legitimate and scanning network traffic, we show that establishing a GTR of scanners based on binary classification can be misleading. In the proposed approach for establishing a continuous GTR (rather than a binary classifier of benign/scanner), remote hosts of a dataset are assigned scores according to a set of behavioural heuristics to identify normal and abnormal activity. We present a GTR metric to measure the distance between the GTR and the evaluated detector, which we argue provide a better evaluation of the detector accuracy in the absence of AGT.

In addition to evaluating scan detectors, the GTR of scanners provided by this methodology can be beneficial in exploring the distribution, patterns, and characteristics of scanners over the Internet. It can also be used as an initial phase to identify scanners that can be further analyzed to detect coordinated scans. While the focus of this chapter is on the problem of scan detection, we believe that the analysis and guidelines for performing an evaluation with real-world network traffic in the absence of absolute ground truth of intrusions, apply to broader problems in the network intrusion detection domain.

Chapter 7

Defending Online Password Guessing Attacks[1]

Brute force and dictionary attacks on password-only remote login services are now widespread and ever increasing. Enabling convenient login for legitimate users while preventing such attacks is a difficult problem. Automated Turing Tests (ATTs) continue to be an effective, easy-to-deploy approach to identify automated malicious login attempts with reasonable cost of inconvenience to users. In this chapter we discuss the inadequacy of existing and proposed login protocols designed to address large-scale online dictionary attacks (e.g., from a botnet of hundreds of thousands of nodes). We propose a new Password Guessing Resistant Protocol (*PGRP*), derived upon revisiting prior proposals designed to restrict such attacks. While PGRP limits the total number of login attempts from unknown remote hosts to as low as a single attempt per username, legitimate users in most cases (e.g., when attempts are made from *known*, frequently-used machines) can make several failed login attempts before being challenged with an ATT. We analyze the performance of PGRP with two real-world datasets and find it more promising than existing proposals.

This chapter is largely independent of the other chapters of this thesis which are largely focused on scan detection. Although password guessing attacks might be considered an escalation of privilege rather than just a reconnaissance activity, the commonality is that the problem focus is similar in malicious remote hosts attempting to compromise the target network. Also, both network scanning and automated password guessing attacks usually adhere to network protocol specifications (both syntactically and semantically), involve large number of queries from the same remote hosts, and aim to collect reconnaissance information that might be utilized in subsequent attacks.

[1]The work in this chapter appeared on a published paper [8] co-authored by M. Mannan. Special thanks for M. Mannan for leading the usability analysis (Section 7.4.2).

7.1 Introduction and Motivation

Online guessing attacks on password-based systems are inevitable and commonly observed against web applications and SSH logins. In a recent report, SANS [94] identified password guessing attacks on websites as a top cyber security risk. As an example of SSH password-guessing attacks, one experimental Linux honeypot setup has been reported [86] to suffer on average 2,805 SSH malicious login attempts per computer per day (see also [38]). Interestingly, SSH servers that disallow standard password authentication may also suffer guessing attacks, e.g., through the exploitation of a lesser known/used SSH server configuration called *keyboard interactive* authentication [93]. However, online attacks have some inherent disadvantages compared to offline attacks: attacking machines must engage in an interactive protocol, thus allowing easier detection; and in most cases, attackers can try only limited number of guesses from a single machine before being locked-out, delayed, or challenged to answer Automated Turing Tests (ATTs, e.g., CAPTCHAs [117]). Consequently, attackers often must employ a large number of machines to avoid detection or lock-out. On the other hand, as users generally choose common and relatively weak passwords (thus allowing effective password dictionaries [74, 119]), and attackers currently control large botnets (e.g., Conficker [77]), online attacks are much easier than before.

One effective defense against automated online password guessing attacks is to restrict the number of failed trials without ATTs to a very small number (e.g., three), limiting automated programs (or bots) as used by attackers to three free password guesses for a targeted account, even if different machines from a botnet are used. However, this inconveniences the legitimate user who then must answer an ATT on the next login attempt.

Several other techniques are deployed in practice, including: allowing login attempts without ATTs from a different machine, when a certain number of failed attempts occur from a given machine; allowing more attempts without ATTs after a timeout period; and time-limited account locking. Many existing techniques and proposals involve ATTs, with the underlying assumption that these challenges are sufficiently difficult for bots and easy for most people. However, users increasingly dislike ATTs as these are perceived as an (unnecessary) extra step; see Yan and

Ahmad [128] for usability issues related to commonly used CAPTCHAs. Due to successful attacks which break ATTs without human solvers (e.g., [127, 112]), ATTs perceived to be more difficult for bots are being deployed. As a consequence of this arms-race, present-day ATTs are becoming increasingly difficult for human users [17], fueling a growing tension between security and usability of ATTs. Therefore, we focus on reducing user annoyance by challenging users with fewer ATTs, while at the same time subjecting bot logins to more ATTs, to drive up the economic cost to attackers [68].

Two well-known proposals for limiting online guessing attacks using ATTs are Pinkas and Sander [84] (herein denoted PS), and van Oorschot and Stubblebine [113] (herein denoted VS). For convenience, a review of these protocols is given in Section 7.6. The PS proposal reduces the number of ATTs sent to legitimate users, but at some meaningful loss of security; for example, in an example setup (with $p = 0.05$, the fraction of incorrect login attempts requiring an ATT) PS allows attackers to eliminate 95% of the password space without answering any ATTs. The VS proposal reduces this but at a significant cost to usability; for example, VS may require all users to answer ATTs in certain circumstances (see Section 7.6). The proposal in the present chapter, called Password Guessing Resistant Protocol (*PGRP*), significantly improves the security-usability trade-off, and can be more generally deployed beyond browser-based authentication.

PGRP builds on these two previous proposals. In particular, to limit attackers in control of a large botnet (e.g., comprising hundreds of thousands of bots), PGRP enforces ATTs after a few (e.g., three) failed login attempts are made from *unknown* machines. On the other hand, PGRP allows a high number (e.g., 30) of failed attempts from *known* machines without answering any ATTs. We define known machines as those from which a successful login has occurred within a fixed period of time. These are identified by their IP addresses saved on the login server as a white-list, or (as in PS [84]) cookies stored on client machines. A white-listed IP address and/or client cookie expire after a certain time.

PGRP accommodates both graphical user interfaces (e.g., browser-based logins) and character-based interfaces (e.g., SSH logins), while the previous protocols deal

exclusively with the former, requiring the use of browser cookies. PGRP uses either cookies or IP addresses, or both for tracking legitimate users. Tracking users through their IP addresses also allows PGRP to increase the number of ATTs for password guessing attacks and meanwhile to decrease the number of ATTs for legitimate login attempts. Although NATs and web proxies may (slightly) reduce the utility of IP address information, in practice, the use of IP addresses for client identification appears feasible [19]. In recent years, the trend of logging in to online accounts through multiple personal devices (e.g., PCs, laptops, smart-phones) is growing. When used from a home environment, these devices often share a single public IP address (i.e., a simple NAT address) which makes IP-based history tracking more user-friendly than cookies. For example, cookies must be stored, albeit transparently to the user, in all devices used for login.

Contributions.

1. Strict but User-friendly ATT-based Scheme: The proposed PGRP scheme is more restrictive against attackers than commonly used counter-measures and two earlier proposals [84, 113]. At the same time, PGRP requires answering fewer ATTs for all legitimate users, including those who occasionally require multiple attempts to recall a password.

2. First Reported Empirical Analysis of ATT-based Schemes: We compare PGRP's performance and usability (e.g., the number of ATTs triggered, ATTs sent to legitimate users) to previous such schemes, using two datasets from a university environment (SSH and web-email login data, covering more than a year).

3. Applicability to Web and Text Logins: PGRP is not limited to web-only login (unlike proposals solely relying on browser cookies), as it uses IP address and/or other methods to identify a remote machine in addition to optionally using cookies. By using text-based ATTs (e.g., textcaptcha.com), SSH login can be adapted to use PGRP.

Organization. Section 7.2 discusses related work on prevention techniques for online dictionary attacks. Section 7.3 presents the PGRP login protocol. Section 7.4 compares PGRP with other ATT-based protocols in terms of security (Section 7.4.1), usability (Section 7.4.2), and required computational resources (Section 7.4.3). A summary of limitations comparing these protocols is given in Section 7.4.4. In Section 7.5, we evaluate PGRP and other ATT-based protocols on two different remote login datasets and analyze the results. Section 7.6 provides a review of the PS and VS protocols. Section 7.7 concludes.

7.2 Related Work

Although online password guessing attacks have been known since the early days of the Internet, there is little academic literature on prevention techniques. Account locking is a customary mechanism to prevent an adversary from attempting multiple passwords for a particular username. Although locking is generally temporary, the adversary can mount a DoS attack by making enough failed login attempts to lock a particular account. Delaying server response after receiving user credentials, whether the password is correct or incorrect, prevents the adversary from attempting a large number of passwords in a reasonable amount of time for a particular username. However, for adversaries with access to a large number of machines (e.g., a botnet), this mechanism is ineffective. Similarly, prevention techniques that rely on requesting the user machine to perform extra nontrivial computation prior to replying to the entered credentials are not effective with such adversaries. Adams et al. [5] proposed a sliding window approach to slow down password brute force attacks including those in which attackers use common passwords to guess the usernames. Exceeding one of a set of optional thresholds (e.g., number of usernames attempted for the same password) results in denying the authentication (e.g., for a particular username) for a configurable period of time.

As discussed in Section 7.1, ATT challenges are used in some login protocols to prevent automated programs from brute force and dictionary attacks. Pinkas and Sander [84] presented a login protocol (PS protocol) based on ATTs to protect against online password guessing attacks. It reduces the number of ATTs that legitimate users

must correctly answer so that a user with a valid browser cookie (indicating that the user has previously logged in successfully) will rarely be prompted to answer an ATT. A deterministic function ($AskATT()$) of the entered user credentials is used to decide whether to ask the user an ATT. To improve the security of the PS protocol, van Oorschot and Stubblebine [113] suggested a modified protocol in which ATTs are always required once the number of failed login attempts for a particular username exceeds a threshold; other modifications were introduced to reduce the effects of cookie theft.

For both PS and VS protocols, the decision function $AskATT()$ requires careful design. He and Han [39] pointed out that a poor design of this function may make the login protocol vulnerable to attacks such as the "known function attack" (e.g., if a simple cryptographic hash function of the username and the password is used as $AskATT()$) and "changed password attack" (i.e., an adversary mounts a dictionary attack before and after a password change event initiated by a valid user). The authors proposed a secure non-deterministic keyed hash function as $AskATT()$ so that each username is associated with one key that should be changed whenever the corresponding password is changed. The proposed function requires extra server-side storage per username and at least one cryptographic hash operation per login attempt.

7.3 Password Guessing Resistant Protocol (PGRP)

In this section, we present the PGRP protocol, including the goals and design choices.

7.3.1 Goals, Operational Assumptions, and Overview

Protocol goals. Our objectives for PGRP include the following:

1. The login protocol should make brute-force and dictionary attacks ineffective even for adversaries with access to large botnets (i.e., capable of launching the attack from many remote hosts).

2. The protocol should not have any significant impact on usability (user convenience). For example: for legitimate users, any additional steps besides entering

login credentials should be minimal. Increasing the security of the protocol must have minimal effect in decreasing the login usability.

3. The protocol should be easy to deploy and scalable, requiring minimum computational resources in terms of memory, processing time, and disk space.

Assumptions. We assume that adversaries can solve a small percentage of ATTs, e.g., through automated programs, brute force mechanisms, and low paid workers (e.g., Amazon Mechanical Turk [1]). Incidents of attackers using IP addresses of known machines for targeted password guessing are also assumed to be minimal. Traditional password-based authentication is not suitable for any untrusted environment (e.g., a keylogger may record all keystrokes, including passwords in a system, and forward those to a remote attacker). We do not prevent existing such attacks in untrusted environments, and thus essentially assume any machines that legitimate users use for login are trustworthy. The data integrity of cookies must be protected (e.g., by a MAC using a key known only to the login server [84]).

Overview. The general idea behind PGRP (see Algorithm 6) is that except for the following two cases, all remote hosts must correctly answer an ATT challenge prior to being informed whether access is granted or the login attempt is unsuccessful: (i) when the number of failed login attempts for a given username is very small; and (ii) when the remote host has successfully logged in using the same username in the past (however, such a host must pass an ATT challenge if it generates more failed login attempts than a pre-specified threshold).

In contrast to previous protocols, PGRP uses either IP addresses, cookies, or both to identify machines from which users have been successfully authenticated. The decision to require an ATT challenge upon receiving incorrect credentials is based on the received cookie (if any) and/or the remote host's IP address. In addition, if the number of failed login attempts for a specific username is below a threshold, the user is not required to answer an ATT challenge even if the login attempt is from a new machine for the first time (whether the provided username-password pair is correct or incorrect). Section 7.3.3 below discusses these differences in further detail.

7.3.2 Data Structure and Function Description

Data structures. PGRP maintains three data structures:

1. W: A list of {source IP address, username} pairs such that for each pair, a successful login from the source IP address has been initiated for the username previously.

2. FT: Each entry in this table represents the number of failed login attempts for a valid username, un. A maximum of k_2 failed login attempts are recorded. Accessing a non-existing index returns 0.

3. FS: Each entry in this table represents the number of failed login attempts for each pair of ($srcIP$, un). Here, $srcIP$ is the IP address for a host in W or a host with a valid cookie, and un is a valid username attempted from $srcIP$. A maximum of k_1 failed login attempts are recorded; crossing this threshold may mandate passing an ATT (e.g., depending on $FT[un]$). An entry is set to 0 after a successful login attempt. Accessing a non-existing index returns 0.

Each entry in W, FT, and FS has a "write-expiry" interval such that the entry is deleted when the given period of time (t_1, t_2, or t_3) has lapsed since the last time the entry was inserted or modified. There are different ways to implement write-expiry intervals (e.g., hashbelt [75]). A simple approach is to store a timestamp of the insertion time with each entry such that the timestamp is updated whenever the entry is modified. At anytime the entry is accessed, if the delta between the access time and the entry timestamp is greater than the data structure write-expiry interval (i.e., t_1, t_2, or t_3), the entry is deleted.

Functions. PGRP uses the following functions (IN denotes input and OUT denotes output):

a) *ReadCredential*`(OUT: un,pw,cookie)`: Shows a login prompt to the user and returns the entered username and password, and the cookie received from the user's browser (if any).

Algorithm 6: PGRP: Password Guessing Resistant Protocol

Input:
`//The keyword 'def' denotes the default parameter value and 'd' denotes day,` $k_1, k_2 \geq 0$
t_1 (def=30d), t_2 (def=1d), t_3 (def=1d), k_1 (def=30), k_2 (def=3)
un, pw, $cookie$ `//username, password, and remote host's browser cookie if any`
W (global variable, expires after t_1)[2] `//whitelist of IP addresses with successful login`
FT (global variable, def=0, expires after t_2) `//table of number of failed logins per username`
FS (global variable, def=0, expires after t_3) `//table of number of failed logins indexed by` srcIP/un for hosts in W or hosts with valid cookies

```
1  begin
2      ReadCredential(un, pw, cookie)          // login prompt to enter username/password pair
3      if LoginCorrect(un, pw) then                    // username/password pair is correct
4          if
           (((Valid(cookie, un,k1,true) ∨ ((srcIP, un) ∈ W)) ∧ (FS[srcIP, un] < k1)) ∨ (FT[un] < k2))
           then
5              FS[srcIP, un] ⇐ 0
6              Add srcIP to W                    // add source IP address to the white list
7              GrantAccess(un, cookie) // this function also sends the cookie if applicable
8          else
9              if (ATTChallenge() = Pass) then
10                 FS[srcIP, un] ⇐ 0
11                 Add srcIP to W
12                 GrantAccess(un, cookie)
13             else
14                 Message('The answer to the ATT challenge is incorrect')
15     else                                          // username/password pair is incorrect
16         if ((Valid(cookie, un,k1,false) ∨ ((srcIP, un) ∈ W)) ∧ (FS[srcIP, un] < k1)) then
17             FS[srcIP, un] ⇐ FS[srcIP, un] + 1
18             Message('The username or password is incorrect')
19         else if (ValidUsername(un) ∧ (FT[un] < k2)) then
20             FT[un] ⇐ FT[un] + 1
21             Message('The username or password is incorrect')
22         else
23             if (ATTChallenge() = Pass) then
24                 Message('The username or password is incorrect')
25             else
26                 Message('The answer to the ATT challenge is incorrect')
27 end
```

b) *LoginCorrect*(`IN: un,pw; OUT: true/false`): If the provided username-password pair is valid, the function returns true; otherwise, it returns false.

c) *GrantAccess*(`IN: un,cookie`): The function sends the cookie to the user's browser and then enables access to the specified user account.

d) *Message*(`IN: text`): Shows a text message.

e) *ATTChallenge*(`OUT: Pass/Fail`): Challenges the user with an ATT and returns "Pass" if the answer is correct; otherwise, it returns "Fail".

[2]For an explanation of the use of expiry intervals, see Section 7.3.2 under "Data structures".

f) $ValidUsername$(IN: un; OUT: true/false): If the provided username exists in the login system, the function returns true; otherwise, it returns false.

g) $Valid$(IN: cookie,un,k_1,state; OUT: cookie,true/false): First, the function checks the validity of the cookie (if any) where it is considered invalid in the following cases: (1) the login username does not match the cookie username; (2) the cookie is expired; or (3) the cookie counter is equal to or greater than k_1. The function returns true only when a valid cookie is received. If state = true (i.e., the entered user credentials are correct, as in line 4 of Algorithm 6), a new cookie is created (if cookies are supported in the login system) including the following information: username, expiry date, and a counter of the number of failed login attempts (since the last successful login; initialized to 0). Notice that if state = true, the function does not send the created cookie to the user's browser. Rather, the cookie is sent later by the $GrantAccess()$ function. If state = false (i.e., the entered user credentials are incorrect, as in line 16 of Algorithm 6) and a valid cookie is received, the cookie counter is incremented by one and the cookie is sent back to the user's browser. No action is performed for all the other cases.

7.3.3 Cookies vs. Source IP Addresses

Similar to the previous protocols, PGRP keeps track of user machines from which successful logins have been initiated previously. Browser cookies seem a good choice for this purpose if the login server offers a web-based interface. Typically, if no cookie is sent by the user browser to the login server, the server sends a cookie to the browser after a successful login to identify the user on the next login attempt. However, if the user uses multiple browsers or more than one OS on the same machine, the login server will be unable to identify the user in all cases. Cookies may also be deleted by users, or automatically as enabled by the private browsing mode of most modern browsers. Moreover, cookie theft (e.g., through session hijacking) might enable an adversary to impersonate a user who has been successfully authenticated in the past [31]. In addition, using cookies requires a browser interface (which, e.g., is not applicable to SSH).

Alternatively, a user machine can be identified by the source IP address. Relying on source IP addresses to trace users may result in inaccurate identification for various reasons, including: (i) the same machine might be assigned different IP addresses over time (e.g., through the network DHCP server and dial-up Internet); and (ii) a group of machines might be represented by a smaller number or even a single Internet-addressable IP address if a NAT mechanism is in place. However, most NATs serve few hosts and DHCPs usually rotate IP addresses on the order of several days [19] (also, techniques to identify machines behind a NAT exist, e.g., [16, 52]).

Drawbacks of identifying a user by means of either a browser cookie or a source IP address include: (i) failing to identify a machine from which the user has authenticated successfully in the past; and (ii) wrongly identifying a machine the user has not authenticated before. Case (i) decreases usability since the user might be asked to answer an ATT challenge for both correct and incorrect login credentials. On the other hand, case (ii) affects security since some users/attackers may not be asked to answer an ATT challenge even though they have not logged in successfully from those machines in the past. However, the probability of launching a dictionary or brute force attack from these machines appears to be low. First, for identification through cookies, a directed attack to steal users' cookies is required by an adversary. Even for a stolen cookie (e.g., through some cross-site scripting attacks [51]), the cookie is considered invalid after k_1 attempts (see the definition of the function $Valid$ in Section 7.3.2). Second, for identification through IP addresses, the adversary must have access to a machine in the same subnet as the user.

Consequently, we choose to use both browser cookies and source IP address (or only one of them if the other is not applicable) in PGRP to minimize user inconvenience during the login process. Also, by using IP addresses only, PGRP can be used in character-based login interfaces such as SSH. An SSH server can be adapted to use PGRP using text-based ATTs (e.g., `textcaptcha.com`). For example, a prototype of a text-based CAPTCHA for SSH is available as a source code patch for OpenSSH [73].

The security implications of mistakenly treating a machine as one that a user

	Strawman Protocol [84]	PS [84]	VS [113]		PGRP Protocol
			Owner	Non-owner	
Q1	0	$N - pN$	$(1-p)b_2$	$max(b_1, (1-p)b_2)$	k_2
Q2	$\frac{1}{2}N$	$\frac{1}{2}pN$	$\frac{1}{2}(N-(1-p)b_2) \approx \frac{1}{2}N$	$\frac{1}{2}(N - max(b_1, (1-p)b_2))$	$\frac{1}{2}(N-k_2)$
Q3	0	0	0	b_1/N	k_2/N
Q4	c/N	c/pN	$\leq min(\frac{c}{p}, b_2+c)/N$	$(b_1+c)/N$	$(k_2+c)/N$

Table 7.1: Comparative security analysis for single-account attacks. (Consider $k_2 = 3$, $p = 0.05$, $b_1 = 5$, and $b_2 = 5$, for concreteness; see Section 7.6 for a review of the PS and VS algorithms)

has previously successfully logged in from is limited by a threshold such that after a specific number of failed login attempts (k_1 in Algorithm 6), an ATT challenge is imposed. For identification through a source IP address, the condition $FS[srcIP, un] < k_1$ in line 4 (for correct credentials) and in line 16 (for incorrect credentials) limits the number of failed login attempts an identified user can make without answering ATTs (see Algorithm 6). Also, as explained in Section 7.3.2, the function $Valid(cookie,un,k_1,true)$ in line 4 updates a counter in the received cookie in which the cookie is considered invalid once this counter hits or exceeds k_1. This function is also called in line 16 to check this counter in case of a failed login attempt.

7.3.4 Decision Function for Requesting ATTs

Below we discuss issues related to ATT challenges as provided by the login server in Algorithm 6. The decision to challenge the user with an ATT depends on two factors: (i) whether the user has authenticated successfully from the same machine previously; and (ii) the total number of failed login attempts for a specific user account. For definitions of W, FT, and FS, see Section 7.3.2.

Username-password pair is valid. As in the condition in line 4, upon entering a correct username-password pair, the user will not be asked to answer an ATT challenge in the following cases:

1. a valid cookie is received from the user machine (i.e., the function $Valid$ returns true) and the number of failed login attempts from the user machine's IP address for that username, $FS[srcIP, un]$, is less than k_1 over a time period determined by t_3;

2. the user machine's IP address is in the whitelist W and the number of failed login attempts from this IP address for that username, $FS[srcIP, un]$, is less than k_1 over a time period determined by t_3;

3. the number of failed login attempts from any machine for that username, $FT[un]$, is below a threshold k_2 over a time period determined by t_2.

The last case enables a user who tries to log in from a new machine/IP address for the first time before k_2 is reached to proceed without an ATT. However, if the number of failed login attempts for the username exceeds the threshold k_2 (default 3), this might indicate a guessing attack and hence the user must pass an ATT challenge.

Username-password pair is invalid. Upon entering an incorrect username-password pair, the user will not be asked to answer an ATT challenge in the following cases:

1. a valid cookie is received from the user machine (i.e., the function $Valid$ returns true) and the number of failed login attempts from the user machine's IP address for that username, $FS[srcIP, un]$, is less than k_1 (line 16) over a time period determined by t_3;

2. the user machine's IP address is in the whitelist W and the number of failed login attempts from this IP address for that username, $FS[srcIP, un]$, is less than k_1 (line 16) over a time period determined by t_3;

3. the username is valid and the number of failed login attempts (from any machine) for that username, $FT[un]$, is below a threshold k_2 (line 19) over a time period determined by t_2.

A failed login attempt from a user with a valid cookie or in the whitelist W will not increase the total number of failed login attempts in the FT table since it is expected that legitimate users may potentially forget or mistype their password (line 16-18). Nevertheless, if the user machine is identified by a cookie, a corresponding counter of the failed login attempts in the cookie will be updated. In addition, the FS entry indexed by the {source IP address, username} pair will also be incremented (line 17). Once the cookie counter or the corresponding FS entry hits or exceeds the threshold k_1 (default value 30), the user must correctly answer an ATT challenge.

	Strawman Protocol [84]	PS [84]	VS [113]		PGRP Protocol
			Owner	Non-owner	
Q1	0	0	0	mb_1/N	mk_2/N
Q2	c/N	c/pN	$\leq min(\frac{c}{p}, b_2 + c)/N$	$(mb_1 + c)/N$	$(mk_2 + c)/N$

Table 7.2: Comparative security analysis for multi-account attacks

Output messages. PGRP shows different messages in case of incorrect {username, password} pair (lines 21 and 24) and incorrect answer to the given ATT challenge (lines 14 and 26). While showing a human that the entered {username, password} pair is incorrect, an automated program unwilling to answer the ATT challenge cannot confirm whether it is the pair or the ATT that was incorrect. However, while this is more convenient for legitimate users, it gives more information to the attacker about the answered ATTs. PGRP can be modified to display only one message in lines 14, 21, 24, and 26 (e.g., "login fails" as in the PS and VS protocols) to prevent such information leakage.

Why not to black-list offending IP addresses. We choose not to create a blacklist for IP addresses making many failed login attempts for the following reasons: (i) this list may consume considerable memory; (ii) legitimate users from blacklisted IP addresses could be blocked (e.g., using compromised machines); and (iii) hosts using dynamic IP addresses seem more attractive targets (compared to hosts with static IP addresses) for adversaries to launch their attacks from (e.g., spammers [126]).

If the cookie mechanism is not available for the login server, PGRP can operate by using only source IP addresses to keep track of user machines. Security and usability implications in this case are discussed in Section 7.4.

7.4 Comparison with other ATT-based Protocols

In this section, we analyze the security, usability, and required system resources of PGRP as compared to a strawman protocol and the PS and VS protocols (see Algorithm 7, 8, and 9 in Section 7.6 for a review of these protocols). This section also provides a comparative summary of major limitations in each protocol.

7.4.1 Security Analysis

Following the previous analysis of PS [84], assume a fixed password space of cardinality N, assume passwords are equi-probable, and that the delay between when the {username, password} pair is entered and the ATT challenge is presented to the user is identical whether or not the credentials are correct. Also assume that adversaries using legitimate users' IP addresses[3] occur rarely.

Single-Account Attacks

In a single account attack, a specific user account is targeted. Following the security analysis of VS [113] in this case, we consider the following questions:

- Q1: What is the expected number of passwords that an adversary can eliminate from the password space without answering any ATT challenge?
- Q2: What is the expected number of ATT challenges an adversary must answer to correctly guess a password?
- Q3: What is the probability of a confirmed correct guess for an adversary unwilling to answer any ATT?
- Q4: What is the probability of a confirmed correct guess for an adversary willing to answer c ATTs?

Table 7.1 compares PGRP with the PS and VS protocols. For simplicity, we use only the case $c \geq 2$ in Q4 for the VS protocol. The answer to Q1 depends on the threshold k_2. The adversary can eliminate only k_2 passwords without answering ATTs. Likewise, for Q2, the expected number of ATTs the adversary must answer to correctly guess a password is one-half of the remaining passwords of the password space after subtracting the number of login attempts that do not require ATTs. Using a small value for k_2 yields $\frac{1}{2}(N - k_2) \approx \frac{1}{2}N$. For Q3, given that k_2 is intended to be small (e.g., 3), the probability of guessing a password for a single-account attack

[3]For example, in case of dynamic IP addresses, an attacker machine may be assigned an IP address previously used by a targeted user's machine.

without answering any ATT is very small (e.g, for 8-char case-sensitive alphabetical passwords chosen randomly, $N = 52^8$ and p(guessing the right password) $= 2/52^8$). For Q4, the adversary has only k_2 free attempts after which ATTs must be answered. Therefore, he can guess a total of $k_2 + c$ passwords with a probability of $(k_2 + c)/N$ to find a correct password.

This analysis shows that PGRP provides improved security over PS and VS with respect to all four questions, and identical security compared to the strawman protocol for $k_2 = 0$.

Multi-Account Attacks

In contrast to a directed attack on a single account, an adversary could attempt to break into multiple accounts at the same time. In fact, this is the current trend of brute force and dictionary attacks [94]. In this case, the adversary usually has access to a large number of machines (e.g., compromised machines in a botnet) and initiates the attack from many sources at the same time. This typically gives the adversary a greater chance of compromising user accounts than targeting a single account.

We compare previous protocols and PGRP by answering the following questions in Table 7.2:

- Q1: What is the probability that an adversary knowing m usernames can correctly guess a password without answering any ATT challenge?
- Q2: What is the probability of a confirmed correct guess for an adversary knowing m usernames and willing to answer c ATTs?

Considering Q1 in Table 7.2, PS appears more secure for multi-account attacks than VS and PGRP. However, it may be unrealistic to assume that an adversary with access to a large botnet is unable to break a small percentage of ATTs [127, 112] (which leads us to Q2).

For Q2, the probability in PGRP depends on the number of usernames the adversary knows and k_2. While PGRP is comparable to the VS protocol in multi-account attacks, PS seems slightly better than PGRP but only for login systems with a large number of users as in equation (7.1).

$$\begin{aligned} \frac{m \cdot k_2 + c}{N} &> \frac{c}{p \cdot N} \\ m &> \frac{c}{k_2}(\frac{1}{p} - 1) \end{aligned} \tag{7.1}$$

To consider a concrete example, for any password length, $k_2 = 3$, $p = 0.05$, and an adversary willing to answer $c = 2^{20}$ ATTs: $m > 1/3\ (2^{20}/0.05 - 2^{20})$; i.e., when $m > 2^{22}$ users, PS is better than PGRP in Q2.

In the PGRP protocol, an adversary may be able to guess a subset of the valid usernames which is undesirable in certain cases [29]. In line 19 of Algorithm 6, the FT list is not updated if the username is invalid, thus an ATT will be requested for each login attempt with an invalid username. Therefore, the adversary could generate a list of valid usernames as follows: if an attempted username requires an ATT for the first login attempt, the username is considered invalid; otherwise, the username is valid. However, the adversary will overlook valid usernames that have at least k_2 failed attempts. While the condition $ValidUsername(un)$ in line 19 can be omitted to overcome this drawback, the number of entries in the list FT will be now proportional to the number of all attempted usernames (whether valid or invalid) by users/attackers within a time period determined by t_2 (see Section 7.3.2 under "Data structures"). We choose to keep the condition $ValidUsername(un)$ in line 19 to restrict the maximum size of FT to the number of valid usernames, even when guessing attacks involving a large number of usernames (both valid and invalid) are launched.

A class of automated password guessing attacks attempts to guess usernames using known common passwords. If there is no mechanism in place to identify and deny common and weak passwords at the time of creating a new account, then this class of attacks might be effective against a portion of accounts with this kind of passwords. Note that PGRP significantly limits such attacks as the maximum number of passwords that the attacker can attempt for any username without answering ATTs is k_2 (3 attempts by default) within t_2 (1 day by default) time window, regardless of the number of remote hosts making the login attempts (lines 19-26 in Algorithm 7).

7.4.2 Usability Comments on ATT Challenges

Our main security goal is to restrict an attacker who is in control of a large botnet from launching online single-account or multi-account password dictionary attacks. In terms of usability, we want to reduce the number of ATTs sent to legitimate users as much as possible. A user receives ATTs when the total number of failed attempts exceeds threshold k_2, and the login attempt is initiated from (i) an unknown machine (i.e., no valid cookies or white-listed IP addresses), or (ii) a known machine from which the user has already failed k_1 times. This happens for both cases of correct and incorrect username-password pairs, assuming the provided username is valid. Below we discuss different login scenarios and the extra effort as required from users by PGRP. The analysis below indicates that only limited usability impact may be expected from our proposal; the same can also be inferred from our real-world data analysis, e.g., the number of ATTs sent to legitimate users (see Section 7.5). However, we have not yet carried out any formal user testing. For notation and parameters as used in the following, see Algorithm 6. For definitions of W, FT, and FS, see "Data structures" in Section 7.3.2.

First time login from an unknown machine. If a valid username-password pair is provided from an unknown machine (i.e., one from which no successful login has occurred within a designated period), no ATTs are required if the total fail count from unknown machines is below k_2 (within a time period determined by t_2). This threshold may be exceeded as follows: (i) the user may provide incorrect passwords from that machine k_2 times; (ii) attackers may have attempted k_2 failed passwords (from unknown machines); or (iii) a combination of (i) and (ii). Once a user successfully logs in, the machine's IP address is added to the known list (W).

Subsequent login from a known machine. ATTs are sent to a known machine (i.e., one from which a successful login has occurred within a designated period) only when k_1 is hit or crossed (see line 4 in Algorithm 6) for that machine and the user account is possibly under attack (i.e., k_2 failed attempts also occurred on the account's username from unknown machines). By setting k_1 to be relatively large (e.g., $k_1 = 30$), legitimate users may make a reasonable number of password mistakes without experiencing any ATTs.

Valid password is provided. Users may be understandably annoyed if they provide a valid password, and yet are asked to answer an ATT. When a valid password is provided by the user, no ATT challenges are sent if the attempt comes from a known machine which has not been used for more than $k_1 - 1$ failed login attempts within a time period determined by t_3. If the user hits or crosses the threshold k_1, still no ATTs are sent if the number of failed login attempts from unknown machines remains below k_2. Thus, users must pass ATT challenges only when they attempt login from unknown machines and the number of failed attempts from unknown machines has hit or crossed k_2 (possibly due to an ongoing attack). We believe this is an uncommon occurrence, as was apparent from our collected data.

Invalid password. This may be a common occurrence for several reasons: (i) if users need multiple attempts to recall the correct password; (ii) if users cycle-through multiple passwords due to multi-password interference [22]; and (iii) typing errors including activating the caps-lock key, sometimes aggravated by on-screen masking of password characters (see e.g., Nielsen's blog [78]). From each known machine, a user is allowed up to k_1 attempts, before challenged with ATTs; i.e., if the user has logged in from n known machines (within a time period determined by t_3), then in total $n \cdot k_1 + k_2$ attempts are allowed without ATTs. While high values of k_1 (30 by default) provide convenient login for legitimate users in common use-cases, we do not recommend very high values (e.g., $k_1 = 10000$) as that may aid guessing attacks when a cookie is stolen or a dynamic white-listed IP address is assigned to an attacker's machine (i.e., a bot). Note that in VS [113], an adversary can make a certain number of failed connection attempts (the threshold b_2 in Algorithm 9) for all (or as many as possible) users of system, with the result that any failed login attempt from a legitimate user will face an ATT challenge. In PGRP, user convenience is unaffected by an attacker's actions, as long as there are not more than $k_1 - 1$ unsuccessful login attempts from known machines.

Invalid username. When a user tries login with a non-existent username (e.g., typing errors), an ATT challenge is given. Irrespective of the password or ATT answer, the login fails. This feature restricts attackers from learning valid usernames (except the usernames obtained via brute force attacks as explained in Section 7.4.1),

and improves protocol performance in terms of memory usage (i.e., no entries in protocol data structures W, FT, or FS). However, from a usability point of view, this is not ideal. We expect that this type of error would be limited in practice (in part because usernames, in contrast to passwords, are echoed on a display).

7.4.3 System Resources

No lists are maintained in the PS protocol (see Algorithm 8), thus no extra memory overhead is imposed on the login server. In the VS protocol (see Algorithm 9), only FT is maintained. The number of entries in this list grows linearly with unique usernames (both valid and invalid) used in failed login attempts. An attacker may try to exhaust a login server's memory by failed login attempts for many usernames. For any cookie-based login protocol, the login server may also need to store information regarding each generated cookie to ameliorate cookie theft attacks [113]. Note that neither the PS nor VS protocol uses IP addresses. The most expensive server operation in PS, VS, and PGRP is generating an ATT.

In PGRP, three tables must be maintained. First, the whitelist, W is expected to grow linearly with the number of users. At any given time, W contains a list of {source IP address, username} pairs that have been successfully authenticated in the last t_1 units of time. Second, the number of entries in FT increases by one whenever a remote host makes a failed login attempt using a valid username, if the username is not already in FT, and the remote host's IP address is not in W (or has no valid cookie). Therefore, unlike the VS protocol, the total number of valid usernames in the login server puts an upper bound on the number of entries in FT since a failed login attempt for a non-existing username does not affect this table.

A new entry is added to FS only when a valid {username, password} pair is provided from an IP address not used before for this username. Therefore, the number of entries in FS is proportional to the number of IP addresses legitimate users successfully authenticated from. Increasing t_3 increases the number of entries in FS since the table entries last longer. The number of entries in FS is expected to be close to the number of active users within the last t_3 units of time (as also shown in the analysis of two real-world datasets in Section 7.5).

		PS [84]	VS [113]	PGRP
Security	Passwords eliminated from the password space of cardinality N	$(\mathbf{1-p})\mathbf{N}$	$(\mathbf{1-p})\mathbf{b_2}$	k_2
	Password space elimination by an adversary with a valid cookie	**N**	**N**	k_1
	Cookie theft	**Yes**	**Yes**	**Yes**
Usability	Probability of ATT for an incorrect password from known machine	**p**	**p**	0 ($a < k_1$) 1 ($a \geq k_1$)
	The attack of making enough failed login attempts for the valid usernames so that legitimate users must then pass ATTs first	No	**Yes**	No
	ATTs for a correct password from unknown machine	Yes	in owner mode	if $a \geq k_2$
	Cookies drawbacks (multiple browsers/machines, deleted cookies)	**Yes**	**Yes**	No
Deployability	AskATT function required	**Yes**	**Yes**	No
	Protocol is suitable for browsers only	**Yes**	**Yes**	No
	Protocol state grows linearly with the number of users	No	**Yes**	**Yes**
	Protocol state grows linearly with usernames in failed attempts	No	**Yes**	No

Table 7.3: Comparison of protocol limitations (limitations are in bold face; a denotes attempts)

7.4.4 Limitations

Table 7.3 summarizes major shortcomings in the PS, VS, and PGRP protocols. Under each protocol, the text is highlighted in bold face if the corresponding entry is a limitation. The first limitation in the security row represents Q1 as discussed in Section 7.4.1. The second limitation is about password space elimination for an adversary with a valid cookie or who can use an IP address from which a username has been successfully authenticated in the past. Neither the PS nor VS protocol restricts the number of failed login attempts for such adversaries. Cookie theft is a possible attack that can be mounted against all these protocols, but can be mitigated by updating a counter in the cookie for the maximum number of failed login attempts [84]. The VS protocol stores cookies only on trustworthy machines (as discussed in Section 7.6), to reduce exposure to cookie theft.

As outlined in Table 7.3, the first limitation in the usability row is about subsequent login from a known machine, as discussed in Section 7.4.2. Only PGRP allows legitimate users to try a relatively large number of wrong passwords ($k_1 = 30$ default)

without passing ATTs. In the second usability limitation, only the VS protocol allows an adversary to make enough failed login attempts for the valid usernames so that legitimate users must then pass ATTs first. The third usability limitation is about ATT challenges for a user who successfully logs in from an unknown machine for the first time (as discussed in Section 7.4.2). Usability drawbacks of cookies are discussed in Section 7.3.3. By using either IP addresses or both cookies and IP addresses for tracking legitimate users, PGRP is the only protocol that avoids usability drawbacks of using cookies.

As discussed in Section 7.2, the design of the deterministic function AskATT() in both PS and VS protocols could have security and deployability drawbacks. Given that the design of both PS and VS protocols considers only cookies to identify machines, only PGRP is designed for both login systems that are web-based and those that are not web-based (e.g., SSH and FTP). The last two limitations in the deployability row are as discussed in Section 7.4.3.

7.5 Empirical Evaluation

In this section we provide the details of our test setup, empirical results, and analysis of PGRP on two different datasets. PGRP results are also compared to those obtained from testing the PS and VS protocols on the same datasets.

7.5.1 Datasets

We used two datasets from an operational university network environment. Each dataset logs events of a particular remote login service, over a one-year period each.

SSH Server Log. The first dataset was a log file for an SSH server serving about 44 user accounts. The SSH server recorded details of each authentication event, including: date, time, authentication status (success, failed, or invalid username), username, source IP address, and source port. Log files were for the period of January 4, 2009 to January 22, 2010 (thus, slightly over one year). Table 7.4 shows that the majority of the login events (95%) are for invalid usernames suggesting that most login attempts are due to SSH guessing attacks. Note that attack login attempts

Number of:	SSH log	Email log
a) Login events	90,190	48,375
i) with valid usernames	5%	99%
ii) with invalid usernames	95%	1%
b) Valid usernames entered	26	147
c) Invalid usernames entered	13,654	130

Table 7.4: Login events from SSH (Jan. 4, 2009 to Jan. 22, 2010) and Horde email servers (Jan. 15, 2009 to Jan. 25, 2010)

involving valid usernames are not distinguishable from incorrect logins by legitimate users since there is no indication whether the source is malicious or benign. However, there were only few failed login attempts for valid usernames either over short bursts or over the whole log capture period. The number of invalid usernames that appear to be mis-typed valid usernames represents less than 1%.

Email Server Log (Web Interface). The second dataset consisted of log files of a Horde IMP email client[4] for the period of January 15, 2009 to January 25, 2010. The Horde email platform is connected to an IMAP email server in a university environment. For each authentication event, a log entry contained: date, time, authentication status (success, failed, or invalid username), username, and source IP address. Although the number of registered user accounts in this server is 1758, only 147 accounts were accessed. Compared to the SSH log, Table 7.4 shows that malicious login attempts are far less prevalent, at only about 1%. Login attempts with valid usernames generated by guessing attacks are, as above, not distinguishable. We were unable to determine the percentage of misspelled valid usernames since the log file data including the usernames was anonymized.

7.5.2 Simulation Method and Assumptions

We performed a series of experiments with a Python-based implementation of PGRP with different settings of the configuration variables (k_1, k_2, t_1, t_2, and t_3). The login events in each dataset are ordered according to date (older entries first). Each event is

[4]Horde IMP is an open source PHP-based Webmail client for IMAP; see http://www.horde.org/imp/.

Exp.	Protocol Settings					Number of ATTs		Entries in W		Entries in FT		Entries in FS	
no.	k_1	k_2	t_1	t_2	t_3	SSH	Email	SSH	Email	SSH	Email	SSH	Email
1	30	0	30	1	1	86,118	6,232	70	524	0	0	12	56
2	30	1	30	1	1	**85,669**	**1,002**	70	524	**1**	**1**	12	56
3	30	2	30	1	1	**85,592**	**728**	70	524	**6**	**9**	12	56
4	30	3	30	1	1	**85,552**	**646**	70	524	6	9	12	56
5	30	4	30	1	1	**85,540**	**617**	70	524	6	9	12	56
6	10	3	30	1	1	**85,552**	**668**	70	524	6	9	12	56
7	30	3	30	2	2	**85,554**	**656**	70	524	6	9	**16**	**79**
8	30	3	10	1	1	**85,552**	**678**	**41**	**219**	6	9	**12**	**56**

Table 7.5: Number of ATTs triggered and number of entries in W, FT, and FS for PGRP (non-default parameters are shaded; for each experiment, changes in results from the previous experiment are in bold face)

processed by PGRP as if it runs in real-time, with protocol tables updated according to the events. Since entries in the tables W, FT, and FS have write-expiry intervals,[5] they get updated at each login event according to the date/time of the current event (i.e., the current time of the protocol is the time of the login event being processed).

We assume that users always answer ATT challenges correctly. While some users will fail in answering some ATTs in practice (see, e.g., [17]), the percentage of failed ATTs depends on the mechanism used to generate the ATTs, the chosen challenge degree of difficulty (if configurable), and the type of the service and its users. The number of generated ATTs by the server can be updated accordingly; for example, if the probability of answering an ATT correctly is p, then the total number of generated ATTs must be multiplied by a factor of $1/p$. Since no browser cookie mechanism was implemented in our tests, in either services of the datasets, the function $Valid(cookie, un, k_1, status)$ always returns false. In the absence of a browser cookie mechanism, a machine from which a user has previously logged in successfully would not be identified by the login system if the machine uses a different IP address that is not in W (see Section 7.3.3 for further discussion). Such legitimate users will be challenged with ATTs in this case.

For a comparative analysis, we also implemented the PS and VS protocols under the same assumptions. The cookie mechanism in these protocols is replaced by IP address tracking of user machines since cookies are not used in either datasets. The

[5]For an explanation of the use of expiry intervals, see Section 7.3.2 under "Data structures".

probability p of the deterministic function (see Section 7.6) is set to 0.05 (suggested by Pinkas and Sander [84]), 0.30, and 0.60 in each experiment. For VS, b_1 and b_2 (see Section 7.6) are both set to 5 (van Oorschot and Stubblebine [113] suggested 10 as an upper bound for both b_1 and b_2).

7.5.3 Analysis of Results

In Table 7.5, we list the protocol parameter settings of 8 experiments. For both SSH and email datasets, the total number of ATTs that would be served over the log period, and the maximum number of entries in the W, FT, and FS tables are reported.

In the first five experiments, we change the parameter k_2 from 0 to 4. k_2 bounds the number of failed login attempts after which an ATT challenge will be triggered for the following login attempt. Note that the total number of ATTs served over the log period decreases slightly with a larger k_2 for both datasets. Other parameters have minor effects on the number of ATTs served.

The number of entries in W in the email dataset is larger than the SSH dataset since there are more email users. Note that although the number of failed login attempts is larger in the SSH dataset, the number of entries in FT is smaller than the email dataset because the number of usernames is less in the SSH dataset with very few common usernames (e.g., common first or last names that can be used in brute force attacks). Given that the protocol requires an ATT for each failed login attempt from a source not in W (and with no valid cookie) when k_2 is set to 0, the FT table is empty in the first experiment for both datasets (as the second condition in line 19 in Algorithm 6 is always false). Increasing t_3 increases the number of entries in FS since the table entries last longer as in the seventh experiment.

Tables 7.6 and 7.7 show the results of the PS, VS, and PGRP protocols for the SSH and email datasets respectively. Configuration variables not listed in the settings columns for PGRP are the default values (as in Algorithm 6). Test results are analyzed from different perspectives below.

a) The number of successful login attempts. The larger the ratio of successful login attempts without answering ATTs to total successful login attempts, the more

Protocol	Settings	Successful Login number of: a) attempts w/ ATT	a) attempts w/o ATT	b) unique usernames w/ ATT	b) unique usernames w/o ATT	Failed Login number of: c) attempts using valid usernames w/ ATT	c) attempts using valid usernames w/o ATT	d) unique valid usernames w/ ATT	d) unique valid usernames w/o ATT	e) attempts using invalid usernames w/ ATT	e) attempts using invalid usernames w/o ATT
PS [84]	$p = 0.05$	346	3,823	24	23	0	563	0	20	3,930	81,528
	$p = 0.30$					146	417	4	16	19,015	66,443
	$p = 0.60$					557	6	18	2	79,408	6,050
VS [113]	$p = 0.05$	346	3,823	24	23	418	145	12	20	50,806	34,652
	$p = 0.30$					444	119	14	16	59,543	25,915
	$p = 0.60$					557	6	18	2	82,160	3,298
PGRP	$k_2 = 0$	412	3,757	24	23	248	315	16	14	85,458	0
	$k_2 = 1$	50	4,119	13	24	161	402	13	20		
	$k_2 = 2$	20	4,149	7	24	114	449	11	20		
	$k_2 = 3$	3	4,166	3	24	91	472	5	20		
	$k_2 = 4$	1	4,168	1	24	81	482	3	20		
	$k_1 = 10$	3	4,166	3	24	91	472	5	20		
	$t_2 = 2, t_3 = 2$	5	4,164	3	24	91	472	5	20		
	$t_1 = 10$	3	4,166	3	24	91	472	5	20		

Table 7.6: Experimental results for the SSH dataset (best results are shaded)

convenient the login experience for the user. For the default parameters of PGRP (i.e., $k_2 = 3$ in Tables 7.6 and 7.7 and other parameters as given in Algorithm 6), the ratio is $4,166/(4,166 + 3) = 0.999$ for the SSH dataset and $46,201/(46,201 + 26) = 0.999$ for the email dataset. The ratio decreases slightly as k_2 is decreased in both datasets. No other parameters significantly affect this ratio. All the experiments have a ratio over 99% except when k_2 is 0 for the email dataset (89%). Both PS and VS protocols have a ratio of $3823/(3823 + 346) = 91\%$ for the SSH dataset and 90% for the email dataset.

b) The number of unique usernames in successful logins. For PGRP default parameters, the number of unique usernames in successful logins that involved answering ATTs (in the SSH dataset) is 3. Thus, the majority of valid users were not challenged with any ATT. For the other dataset, 11 valid usernames (out of 147) faced an ATT challenge. Almost all usernames were used in successful logins without answering ATTs in both datasets. k_2 and t_2 are the only parameters that affected the results. For both datasets, most SSH users were asked to answer ATTs in both the PS and VS protocols; therefore, PGRP offers a more convenient login for legitimate users.

Protocol	Settings	Successful Login number of: a) attempts w/ ATT	a) attempts w/o ATT	b) unique usernames w/ ATT	b) unique usernames w/o ATT	Failed Login number of: c) attempts using valid usernames w/ ATT	c) attempts using valid usernames w/o ATT	d) unique valid usernames w/ ATT	d) unique valid usernames w/o ATT	e) attempts using invalid usernames w/ ATT	e) attempts using invalid usernames w/o ATT
PS [84]	$p = 0.05$	4,609	4,1618	134	102	166	1,408	7	101	24	550
	$p = 0.30$					442	1,132	29	79	85	489
	$p = 0.60$					1,524	50	98	10	543	31
VS [113]	$p = 0.05$	4,609	41,618	134	102	961	613	41	101	291	283
	$p = 0.30$					1,103	471	55	79	350	224
	$p = 0.60$					1,528	46	100	10	545	29
PGRP	$k_2 = 0$	5,283	40,944	134	100	375	1,199	88	71	574	0
	$k_2 = 1$	279	45,948	69	127	149	1,425	47	108		
	$k_2 = 2$	81	46,146	32	132	73	1,501	18	108		
	$k_2 = 3$	26	46,201	11	134	46	1,528	11	108		
	$k_2 = 4$	9	46,218	5	134	34	1,540	6	108		
	$k_1 = 10$	29	46,198	13	134	65	1,509	13	108		
	$t_2 = 2, t_3 = 2$	36	46,191	11	134	46	1,528	11	108		
	$t_1 = 10$	34	46,193	13	134	70	1,504	13	108		

Table 7.7: Experimental results for the email dataset (best results are shaded)

c) The number of failed login attempts with valid usernames. Failed login attempts with valid usernames could be from either malicious or benign sources. In the first experiment on PGRP ($k_2 = 0$), there are 315 failed attempts not involving ATTs in the SSH dataset and 1,199 in the email dataset. Given that the source IP addresses of all these attempts are in W, these failed attempts are considered benign. In general, the lower the number of attempts with ATTs the better for user convenience. For PGRP default parameter settings, 16% $(91/(472+91))$ of the failed attempts (with valid usernames) involved ATT challenges in the SSH dataset and 3% $(46/(1,528+46))$ in the email dataset. Even if we assume that all failed attempts (with ATTs) are made by legitimate users, PGRP results are better compared to 74% $(418/(418+145))$ for the SSH dataset and 61% for the email dataset in the VS best case (for $p = 0.05$). PS offers slightly better results, however, this is only when $p = 0.05$ which also reduces the number of required ATTs for password guessing attempts (i.e., with invalid usernames as in the last column in Tables 7.6 and 7.7).

d) The number of unique valid usernames in failed login attempts. In both datasets, setting $k_2 \geq 1$ in PGRP causes a significant decrease in the number of unique valid usernames that face ATT challenges in failed login attempts. Other

parameters have no significant effect in this manner. For $k_2 = 3$ (default value), in both datasets the number of affected usernames (i.e., the number of legitimate users that are asked to answer ATTs for failed login attempts) is comparable to PS results but less than VS; therefore, PGRP offers a more convenient login for legitimate users.

e) The number of failed login attempts with invalid usernames. Any login attempt with invalid username triggers an ATT in PGRP (i.e., no failed login attempt with invalid usernames avoids an ATT). Indeed, all attempts with invalid usernames trigger ATTs in both datasets. In contrast, for the SSH dataset, only 0.046% in PS and 0.59% in VS trigger ATTs for $p = 0.05$ (0.04% and 0.51% in the email dataset).

Summary of comparison. The trade-off between user convenience (item (c) above) and login security with respect to password guessing (item (e)) in both PS and VS protocols is evident from the above discussion; i.e., increasing the number of ATTs to limit password guessing attempts also increases the number of ATTs legitimate users must answer. Such a trade-off is significantly limited with PGRP. Moreover, the number of legitimate login attempts that trigger ATTs (and the number of affected users) is significantly lower in PGRP than both PS and VS. On the other hand, in PGRP, more ATTs must be answered in password guessing attacks; if g is the number of password guessing attempts for m usernames, PGRP requires answering ATT challenges for at least $g - k_2 m$ password guessing attempts. Our datasets represent two very different scenarios: the SSH server received almost 95% invalid login attempts, and the email server received only 1% of such attempts (see Table 7.4). Yet, as the above analysis indicates, PGRP is significantly better (for both security and usability) than previous ATT-based protocols in both cases, and it can be deployed without affecting the login experience of legitimate users.

7.6 Background on Previous ATT-based Protocols

Pinkas and Sander [84] introduced the topic with a strawman login protocol (see pseudo-code in Algorithm 7) that requires answering an ATT challenge first before entering the {username, password} pair. Failing to answer the ATT correctly prevents the user from proceeding further. This protocol requires the adversary to pass an ATT

challenge for each password guessing attempt, in order to gain information about correctness of the guess.

While this simple protocol is effective against online dictionary attacks, assuming that the used ATTs are secure, legitimate users must also pass an ATT challenge for every login attempt. Therefore, this protocol affects user convenience substantially, and requires the login server to generate an ATT challenge for every login attempt.

Pinkas and Sander [84] then made their actual proposal, a login protocol that reduces the number of ATTs legitimate users are required to pass; see pseudo-code in Algorithm 8 (PS protocol). The protocol stores a browser cookie on the machine of users who had previously logged in successfully. The cookie is tied to the username of the last successful login attempt.

Once the user requests the login server URL, the user's browser sends the cookie (if any) back to the server. The protocol then requests the user to enter a {username, password} pair. If the pair is correct and a valid cookie (i.e., an unexpired cookie indicating that a successful login for the username was made from the same browser) is received from the browser then the user is granted access. If the pair is correct but no valid cookie is received, then an ATT challenge must be answered before account access is granted. Otherwise, if the {username, password} pair is incorrect then according to a function $AskATT(username, password)$, an ATT challenge might be required before informing the user that the {username, password} pair is incorrect.

$AskATT(username, password)$ must be a deterministic function of the entered {username, password} pair such that for a specific pair, an ATT challenge is either always requested, or never (this function is denoted $AskATT(un, pw)$ in Algorithm 8). That is, for a password space of size N, pN of the possible passwords require ATTs (e.g., if $p = 0.05$, $0.05 \times N$ of the password space for a given username require ATTs).

With this protocol, legitimate users must pass ATTs in the following cases: (i) when the user logs in from a machine for the first time; and (ii) when the user's {username, password} pair is incorrect and $AskATT()$ triggers an ATT. On the other hand, an automated program needs to correctly answer an ATT for each password guessing attempt except one case: when the {username, password} pair is incorrect and a deterministic function $AskATT()$ did not request an ATT.

Algorithm 7: Secure but inconvenient login protocol [84]

```
1  begin
2      if ATTChallenge() = Pass then
3          ReadCredential(un, pw)   //login prompt to enter username/password
4          if LoginCorrect(un, pw) then    //username/password pair is correct
5              Access is granted to the account
6          else
7              Message('The username or password is incorrect')
8      else
9          Message('ATT answer is incorrect')
10 end
```

Algorithm 8: PS protocol, adapted from Pinkas and Sander [84]

```
1  begin
2      ReadCredential(un, pw, cookie)      //login prompt
3      if LoginCorrect(un, pw) then        //username/password pair is correct
4          if Valid(cookie, un) then   // cookie unexpired and matches username
5              GrantAccess(un)             // access is granted to the account
6          else                          // no cookie or the cookie is invalid
7              if ATTChallenge() = Pass then
8                  GrantAccess(un)
9              else
10                 Message('login fails')
11     else                              // username/password pair is incorrect
12         if AskATT(un, pw) = True then
13             if ATTChallenge() = Pass then
14                 Message('login fails')
15             else
16                 Message('login fails')
17         else
18             Message('login fails')
19 end
```

In addition to the correct password, this protocol requires ATTs for a fraction p of the incorrect passwords. Therefore, an adversary can confirm that $(1-p)(N-1) \approx N - pN$ of the passwords in the password space N are incorrect without answering any ATT challenge. The expected number of ATTs an adversary must correctly answer to guess a password correctly is $\frac{1}{2}pN$. Thus, if the adversary is willing to answer c ATTs, the probability of finding a correct password is c/pN. For better

Algorithm 9: VS protocol, adapted from van Oorschot and Stubblebine [113]

Input:
FT (global variable, def=0, expires after t_2) `//table of number of failed logins per username`

```
1  begin
2      ReadCredential(un, pw, cookie)                    // login prompt to enter
       username/password pair
3      if LoginCorrect(un, pw) then        // username/password pair is correct
4          if Valid(cookie, un) then   // cookie unexpired and matches username
5              Access is granted to the account
6          else                                // no cookie or the cookie is invalid
7              if (OwnerMode(un)) OR (FT[un] ≥ b1) then
8                  if ATTChallenge() = Pass then GrantAccess(un)  // access is
                   granted to the account
9                  else Message('login fails')
10             else
11                 GrantAccess(un)          // access is granted to the account
12     else                                // username/password pair is incorrect
13         if (AskATT(un, pw) = True) OR (FT[un] ≥ b2) then
14             if ATTChallenge() = Pass then Message('login fails')
15             else Message('login fails')
16         else
17             Message('login fails')
18 end
```

defence against online dictionary attacks, the function $AskATT()$ should request ATTs for the majority of the possible passwords in the overall password space (e.g., $p > 0.75$). However, the probability that a legitimate user is given an ATT challenge upon entering an incorrect password will also increase, creating a trade-off between password security and user convenience. In fact, setting $p = 1$ makes this protocol similar to the strawman protocol, except for successful logins with valid cookies where no ATT is required.

Van Oorschot and Stubblebine [113] proposed modifications to the previous protocol (see Algorithm 9; VS protocol) which track failed logins per username to impose ATT challenges after exceeding a configurable threshold of failures (threshold b_1 for correct {username, password} pair and threshold b_2 for incorrect pair; see Algorithm 9). Hence, for an incorrect {username, password} pair, the decision to request an ATT not only depends on the function $AskATT()$ but also on the number of failed

login attempts for the username (line 13 in Algorithm 9).

In addition, upon entering correct credentials in the absence of a valid cookie, the user is asked whether the machine in use is trustworthy and if the user uses it regularly. The cookie is stored in the user's machine only if the user responds yes to the question. This approach aims to reduce the possibility of cookie theft since a negative answer is expected if the user logs in from a public machine. The user account is set to be in *non-owner mode* for a specified time window when a login is successful without receiving a valid cookie from the user machine; otherwise the account is set to *owner mode*.

The number of incorrect passwords that an adversary can eliminate without passing any ATT challenge is decreased to about $(1-p)b_2$. Moreover, the adversary is expected to need to correctly answer about $N/2$ ATTs in order to guess a password correctly as opposed to $\frac{1}{2}pN$ in the PS protocol. While this VS protocol addresses the security drawback of the PS [84] algorithm, the legitimate user always faces an ATT challenge once the threshold b_2 is exceeded. This feature enables adversaries to affect user login convenience, by initiating $\geq b_2$ failed login attempts for each targeted username, forcing ATT challenges for the subsequent login attempts.

7.7 Concluding Remarks

Online password guessing attacks on password-only systems have been observed for decades (see e.g., [109]). Present-day attackers targeting such systems are empowered by having control of thousand to million-node botnets. In previous ATT-based login protocols, there exists a security-usability trade-off with respect to the number of free failed login attempts (i.e., with no ATTs) versus user login convenience (e.g., less ATTs and other requirements). In contrast, PGRP is more restrictive against brute force and dictionary attacks while safely allowing a large number of free failed attempts for legitimate users. Our empirical experiments on two datasets (of one-year duration) gathered from operational network environments show that while PGRP is apparently more effective in preventing password guessing attacks (without answering ATT challenges), it also offers more convenient login experience, e.g., fewer ATT challenges for legitimate users even if no cookies are available. However, we

reiterate that no user-testing of PGRP has been conducted so far. In addition to our offline empirical evaluation, we believe that testing PGRP on a publicly accessible operational login system with a reasonable number of user accounts will be helpful to further analyze the security and usability of the PGRP protocol.

PGRP appears suitable for organizations of both small and large number of user accounts. The required system resources (e.g., memory space) are linearly proportional to the number of users in a system. PGRP can also be used with remote login services where cookies are not applicable (e.g., SSH and FTP).

Chapter 8

Further Discussion and Conclusion

In this chapter, we provide a comparative summary of the major scanning detection proposals discussed in this thesis, summarize our main results, and revisit the thesis hypotheses. We also describe future research directions.

8.1 Comparison of EM, TRW, LQS, and STRW

In Chapter 3, we performed an analytical and empirical analysis of TRW and EM, two known scan detection algorithms. We have also compared LQS and STRW with TRW in Chapters 4 and 5. We provide a comparative overview of these algorithms in Table 8.1, using the limitations and strengths that we have identified for each algorithm. We give a rate ranging between (★★★★★) for a good performance in the corresponding metric/property (e.g, a high TP rate and a low FP rate both get five filled stars) to (★☆☆☆☆) for a poor performance in the corresponding metric/property. A positive mark (✓) means a property is provided or that the algorithm is resistant to an attack/evasion, and a negative mark (✗) means the lack of that property or that the algorithm is not resistant to an attack/evasion. NA denotes non-applicability.

The metrics in the detection accuracy row are explained in Section 3.3.3. The given scores are the author subjective interpretation and ranking based on the empirical results and the analytical analysis that we have performed throughout Chapters 3, 4, and 5. While we note that these empirical experiments are conducted on specific datasets and network environments as described earlier, based on the analytical analysis, we expect similar results on other datasets according to the discussed network properties. For a given metric, these scores are relative to each other, rather than reflecting the absolute values of the corresponding metrics. k represents the number of failed connection attempts a remote host can initiate (towards newly visited local IP addresses in the monitored network) before being classified as a scanner by

the corresponding algorithm (with the assumption that no successful connections are initiated by the remote between the failed attempts towards newly visited local IP addresses in the case of the TRW and STRW algorithms). k is calculated as a function of the corresponding algorithm default parameters as previously explained. The first property in the essential properties row gives the value of k that results for the default parameter settings for each algorithm.

As we have discussed in Chapter 5, for this value of k, the TRW and STRW algorithms have a similar average TP rate. EM showed the best TP rate, detecting almost all scanners, as it detects scanners from their first connection attempt. In contrast, while LQS has a lower TP rate than EM, it is higher than TRW and STRW (as discussed in Chapter 4). While incrementing k decreases the TP rate of TRW, LQS, and STRW, decrementing k has a minimal effect on TRW and STRW (note that k is constant with a value of one in the case of EM, and $k \geq 2$ in the case of LQS).

EM has the worst FP rate (as discussed in Chapter 3). In comparison with TRW, LQS has a slightly better FP rate than TRW and STRW. However, both LQS and STRW have a significantly lower FP rate than TRW in network environments of transient nature (as discussed in Section 5.1 in Chapter 5). Incrementing k slightly improves the FP rate of TRW, LQS, and STRW, while decrementing k slightly increases the FP rate of these algorithms.

The efficiency metric used in Table 8.1 depends on both true and false positives (the efficiency is the proportion of the reported scanners by the detector that are true positive; see Section 3.3.3) for the definition of the efficiency metric), we find that LQS scores the best efficiency and EM scores the worst efficiency. Incrementing k decreases the efficiency of TRW, LQS, and STRW while decrementing k increases the efficiency of TRW, LQS, and STRW. This effect is mainly linked to the change in the number of true positives.

While all algorithms detect horizontal scans, only EM and LQS detect vertical scans. As discussed above, EM detects scanners from their first connection attempt, and thus EM has the highest detection rate of stealthy scanners, relative to TRW, LQS, and STRW. LQS comes next, as it detects scanners as early as from their second

			EM [123]	TRW [45]	LQS [11]	STRW [9]
Detection Accuracy	TP rate	$k = default$	★★★★★	★★★☆☆	★★★★☆	★★★☆☆
		$k = default + 1$	NA	★★☆☆☆	★★★☆☆	★★☆☆☆
		$k = default - 1$	NA	★★★☆☆	NA	★★★☆☆
	FP rate	$k = default$	★☆☆☆☆	★★★☆☆	★★★★☆	★★★★☆
		$k = default + 1$	NA	★★★☆☆	★★★★☆	★★★★☆
		$k = default - 1$	NA	★★★☆☆	NA	★★★★☆
	Efficiency	$k = default$	★☆☆☆☆	★★★☆☆	★★★★★	★★★★☆
		$k = default + 1$	NA	★★☆☆☆	★★★★☆	★★★☆☆
		$k = default - 1$	NA	★★★☆☆	NA	★★★★☆
Essential Properties	Default value of the threshold k		1	4	2	4
	Detection of horizontal scans		✓	✓	✓	✓
	Detection of vertical scans		✓	✗	✓	✗
	Detection of stealthy scanners		★★★★★	★★★☆☆	★★★★☆	★★★☆☆
	Suitability for detection of rapid worms		★★★★☆	★★☆☆☆	★★★★★	★★★☆☆
	Suitability for environments of transient nature		★☆☆☆☆	★★☆☆☆	★★★☆☆	★★★★☆
	Use efficiency of monitoring-system resources		★★★★★	★★☆☆☆	★★★★☆	★★☆☆☆
Evasion Resistance	TCP retransmission timeout attack		✓	✗	✓	✓
	Using friendly list to bypass detection		✓	✗	✓	✗
	Using known services to delay detection		✓	✗	✓	✗
	Using a large number of remotes to bypass detection		✗	✗	✗	✗
	DoS attack using spoofed remote IP addresses		✗	✗	✗	✗

Table 8.1: Comparison of scan detection algorithms (See Section 8.1 for explanation of check-marks and star ratings)

connection attempt to the monitored network.

In Section 4.4, we have shown that, unlike TRW, LQS does not need to examine the state (i.e., to check whether successful or unsuccessful) of new connection attempts (determining the state of failed connection attempts usually requires a few minutes in most operating systems), and therefore, LQS is more appropriate for detection of rapidly spreading worms than TRW. Although STRW is similar to LQS in not needing to wait for the connection state to make a decision regarding an inbound connection attempt, we give STRW a lower score because of its detection accuracy is

lower than LQS, which is an essential factor in worm detection.

For scan detection in environments of transient nature, we give STRW the highest score because of the following: (i) STRW takes into account various possible causes of benign failed connection attempts (as discussed in Chapter 5); and (ii) only remote hosts that make consecutive failed attempts to four or more internal hosts in the target network will be reported as a scanner. While LQS also has the property (i), LQS is given a lower score as it is more aggressive in detecting scanners (i.e., from the second connection attempt) which might yield more false positives on transient network services. EM is the least appropriate algorithm for such environments because of its high FP rate. From our analysis of these four algorithms, EM is the most efficient algorithm in using system resources. LQS is the second-ranking algorithm and TRW and STRW come in the third rank (see Sections 3.2.6 and 4.4 for further discussion).

As outlined in Table 8.1, the first attack is about taking advantage of the TCP retransmission timeout so that a single remote could send thousands or millions of first packets (e.g., SYN packets) in the duration of a TCP retransmission timeout (a few minutes in most operating systems) to different local hosts and receives responses from open ports in the target network before being detected. As discussed in Section 4.4, only TRW is susceptible to this attack.

Both TRW and STRW use a list of friendly remotes so that 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 further. Therefore, if a local host initiates a connection to a malicious remote, the remote can scan the network without being detected (for further discussion, see Section 4.4).

All four algorithms are susceptible to distributed scans where a set of remote hosts collectively coordinated by the same adversary can divide up the targeted address space among these remotes, or perhaps launch subsequent attacks from remotes not used in the scanning phase. Also, for all the four algorithms, it is feasible for an adversary with limited resources to perform a DoS attack against a target network if a blocking policy is in effect for remotes that are classified as scanners. All four algorithms are susceptible to this attack. An active IDS that tries to complete the TCP 3-way handshake can verify if the remote IP address is spoofed.

8.2 Matching Scan Detector Algorithms to Network Environments

Choosing the appropriate scan detection algorithm depends heavily on the nature of the monitored network environment, security policy, and the importance of these properties/attacks in the deployment environment.

The EM algorithm fits for environments where network security is vital or those where detecting such reconnaissance activity is crucial even at the cost of high false alarm rate. This is because EM detects scanners from their first contact of the monitored network. For common network environments, including both enterprise-level and ISP-level traffic, the high false alarm rate of EM makes it inappropriate.

Unlike EM and LQS, the sequential hypothesis testing model in TRW and STRW provides a theoretical basis for classifying remote hosts. To avoid penalizing benign hosts that make some failed connection attempts, TRW and STRW credit remote hosts that make successful connections by reducing their likelihood ratio towards being classified as benign. This feature enables these two detection algorithms to maintain a reasonable false alarm rate. However, the low false alarm rate comes at the cost of low detection rate as well (relative to the EM and LQS detection algorithms). While decreasing k helps slightly in improving the detection rate, it also increases the FP rate, and there is no publicly available criterion (whether automatically or manually) specifying how to choose appropriate values in TRW to get satisfactory detection results. In general, given the nature of enterprise networks (e.g., restricted and stable publicly accessible network services, and the importance of maintaining a low false alarm rate), we find TRW suitable for enterprise networks where detecting non-stealthy scanners is sufficient.

In contrast to TRW, STRW fits network environments of a transient nature (e.g., wireless networks, residential networks, and some university environments), in which several network hosts and services can be added or stopped (either permanently or temporarily) over time. As we discussed in Chapter 5, in such environments, benign remotes may make failed connection attempts because the services they are trying to connect to, while active in the past, are temporarily unavailable or disabled (because the local host running the service is turned off/sleeping/hibernating, disconnected from the network, or the application running the network service is uninstalled or

closed). These environments include enterprises with a relaxed security policy or fewer restrictions on what applications or services can be used by enterprise workstations, or sometimes with no explicit security policy (e.g., some university networks).

LQS offers both high detection rate and low false positive rate. Given LQS lightweight algorithm, detection speed (in terms of the number of connection attempts that a scanner can perform before being detected), ability to detect stealthy and vertical scans, and resistance to evasion, we believe LQS is appropriate for a wide range of network environments, including enterprise, residential, and ISP environments.

8.3 Revisiting Thesis Objectives

As stated in Chapter 1, the first thesis objective (G1 in Section 1.2) is about gaining a better understanding of the nature, motive, and the current trends of two widespread types of malicious network activity using real-world network traces. This objective is met as follows. First, in our analytical and empirical comparison of the TRW [45] and EM [123] scan detection algorithms in Chapter 3 and our evaluation of the LQS scan detection algorithm in Chapter 4, we have studied the current trend and nature of network scanning events using two recent datasets collected at various sites of different natures and sizes. Also, in Chapter 5, we have conducted several empirical experiments on another two datasets from two qualitatively different network environments. In these four datasets, we have established a reference baseline of scanners to compare against. We find that stealthy scanning activity is now becoming a scanning trend to avoid triggering IDSs. This is now recognized as a feasible and practical strategy due to the availability of large numbers of compromised machines in some coordinated scanning campaigns. Second, in our study and evaluation of the security and usability of two known ATT-based login protocols (see Chapter 6), we have empirically analyzed online password guessing attacks in two datasets from an operational network environment, where each dataset logs events of a particular remote login service, over a one-year period each. The current trend of these guessing attacks seems to be of a large-scale nature targeting apparently random networks. Similar to network scanning, this trend seems linked to the availability of large numbers of compromised machines to some adversaries, which makes searching for easy targets

(e.g., those using default and common passwords) more feasible.

The second thesis objective (G2 in Section 1.2) is about evaluating and comparing existing defense mechanisms, to establish a baseline that can be used to improve the state-of-the-art. For network scanning detection, we have provided an in-depth study (including analytical and empirical evaluation) of two known network scanning detection techniques, the TRW and EM algorithms, in Chapter 3. We have also conducted further experiments on the TRW algorithm in Chapters 4 and 5, using a total of four network traces gathered from four different network environments. Our evaluation confirmed the trade-off between detection and false positive rates in both algorithms and revealed that it is crucial, due to the lack of both a built-in algorithmic adaptability and a manual parameterization criterion based on the properties of the monitored network environment. We also found that manual configuration of parameter choices is hard, especially for non-experts (as we discussed in Sections 3.2.2, 5.2.3, and 5.4). For online password guessing attacks, we have analyzed the security and usability of two well-known ATT-based login protocols: the PS [84] and VS [113] protocols. We are also the first to empirically analyze the performance of these protocols using real-world datasets. Our experiments confirmed that there is a fundamental trade-off between user login convenience and login security with respect to password guessing in existing login protocols; i.e., increasing the number of ATTs to limit password guessing attempts also increases the number of ATTs legitimate users must answer (as identified in previous studies, see [84, 113]).

The third thesis objective (G3 in Section 1.2) is to improve selected existing defensive approaches and to provide new practical defenses based on our analysis of the limitations of the existing approaches and the new trends of network scanning and online password guessing attacks. In Chapter 4, we presented LQS, a new lightweight network scan detection algorithm that detects scanners as early as from their second connection attempt to the monitored network. In our conducted empirical evaluation (as summarized in Table 8.1), LQS exhibited significant improvements over TRW in all the following key properties: (i) fast detection of scanning activity to enable prompt response by IDSs; (ii) acceptable rate of false alarms, keeping in mind that false alarms may lead to legitimate traffic being penalized; (iii) high detection rate

with the ability to detect stealthy scanners; (iv) efficient use of monitoring system resources; and (v) immunity to evasion. We have also presented STRW in Chapter 5, a modified TRW algorithm that utilizes active mapping of network services to take into account benign causes of failed connection attempts. Based on experimental results from a set of tests on two datasets, STRW eliminates a significant portion of TRW false positives. We have shown that while TRW was designed for scan detection in a controlled enterprise network environment, the believed hypothesis that behaviour-based network scanning detectors like TRW exhibit unsatisfactory performance in now common environments of transient nature (e.g., residential style network traffic) [103] is actually due to the lack of utilizing information of the characteristics of the monitored environment. This also shows that the default parameters of such algorithms should not be assumed to give a satisfactory detection accuracy, and they should be configured according to the deployment environment.

In Chapter 7, we presented PGRP, a new password guessing resistant protocol, derived upon revisiting prior proposals designed to restrict online password guessing attacks. While PGRP limits the total number of login attempts from unknown remote hosts to as low as a single attempt per username, legitimate users (e.g., user machines known to the login system) in most cases can make several failed login attempts before being challenged with an ATT. Using two real-world datasets (over a one-year period each), our empirical experiments showed that while PGRP is apparently more effective in preventing password guessing attacks than existing defensive proposals, it also offers more convenient login experience (in terms of fewer ATT challenges for legitimate users even if no cookies are available).

8.4 Revisiting Thesis Hypotheses

The analysis and evaluation of our two new network scan detection algorithms (i.e., LQS and STRW) established that the first part of Hypothesis 1 is confirmed. It states "Incorporating selected properties of mainstream scan detector's operational environment into operational parameters of the scan detector will improve the detection accuracy in terms of true and false positive rates". Both algorithms utilize

the monitored network profile in terms of network services active mapping to distinguish between benign and malicious causes of unsuccessful connection attempts. Thus, this is used in the decision oracle to determine whether a new connection attempt is a part of scanning activity. Indeed, our empirical evaluation has shown that for the datasets and network environments studied, the detection accuracy of both algorithms significantly outperformed the state-of-the-art TRW algorithm.

We have also explored the second part of Hypothesis 1 "It is possible for some scan detection algorithms to automate the process of setting these parameters rather than relying on the network administrator to manually choose appropriate values". Our experiments on four qualitatively different network environments showed that while we can automate few parameters by integrating some extracted data from the deployment environment in a scan detection algorithm, it is sometimes necessary to manually set some parameters mainly because a satisfactory detection accuracy differs according to the cost of false positives and the value of detecting scanners at a given network.

Hypothesis 2 states 'It is possible to design a scan detector that simultaneously provides high detection accuracy, fast detection speed (the speed is measured in terms of the number of connection attempts that a scanner can perform before being detected), and efficient use of monitoring system resources.". As discussed in Chapter 4, we demonstrated that while LQS detects scanners as early as from their second connection attempt to the monitored network, the high detection rate in LQS is not at the cost of high false positive rate. Also, our analysis showed that LQS requires manageable memory footprint even for large networks or high-volumes of network traffic. Therefore, we have established that this hypothesis is confirmed.

By not penalizing remote hosts making benign outbound failed connection attempts as in the proposed STRW algorithm, we were able to significantly reduce false positives (relative to TRW) in all of the datasets that we have experimented on. This supports Hypothesis 3 that states "For a scan detector based on the absolute number of a remote host's successful or failed connection attempts, the false positive rate can be significantly reduced by designing the detector to take into account the various possible causes of benign failed connection attempts which should not be considered

as scanning activity".

Through our various experiments on network environments of different natures, some scan detection algorithms exhibited reasonable detection accuracy on certain network environments but not on others. For example, while TRW scored high detection rate and low false positive rate on the dataset I in Chapter 3, a significant number of the remotes classified as scanners by TRW (19% to 50%, according to the experiment settings) in the first dataset in Chapter 5 are false positives. In contrast, the STRW algorithm exhibits satisfactory performance (in terms of detection accuracy) in network environments of a dynamic or transient nature (e.g., residential style networks), as in the case of the first dataset in Chapter 5. Therefore, we are able to provide support for Hypothesis 4 that states "Some scan detectors are more appropriate for certain types of network environments than others, and we can identify these environments for several scan detectors studied".

In Chapter 6, we showed that establishing absolute ground truth (AGT) of scanners in real-world network traffic is typically infeasible as typically many connection attempts can be equally interpreted as legitimate or scans based on the remote host's intent. We modelled the process of performing an evaluation in the absence of AGT based on our analysis of the problems that can arise when evaluation is based on a GTR rather than AGT. We also set the requirements of using a ground truth reference (GTR) for either evaluating one intrusion detector or comparing multiple detectors. We have shown that a ground truth of scanners that is based on a discrete classification criteria could be misleading and we presented a new evaluation approach for scan detectors designed to address uncertainties in GTR. Accordingly, we have established that Hypothesis 5 is true, which states "There are approaches that can be taken to address and model uncertainties in a ground truth reference (GTR) in the absence of AGT, either in establishing a GTR for a given network trace or in comparing one or more detectors with the GTR".

We were able to provide evidence supporting the first part of Hypothesis 6 that states "It is possible to design a more effective and more general password-based login system with a threat model of large-scale password guessing attacks in mind",

through development and analysis of PGRP, a new password guessing resistant protocol designed with a threat model of large-scale password guessing attacks in mind. PGRP is not limited to web-only login, as it uses IP address and/or other methods to identify a remote machine in addition to optionally using cookies.

Both our empirical evaluation and our security and usability analysis showed that PGRP is more restrictive against attackers than existing defensive proposals (e.g., [84, 113]). While PGRP limits the total number of login attempts from unknown remote hosts to as low as a single attempt per username, making such guessing attacks less effective to adversaries (including large-scale attacks from a botnet of hundreds of thousands of nodes), PGRP requires answering fewer ATTs for all legitimate users, including those who occasionally require multiple attempts to recall a password. Therefore, according to our question in Hypothesis 6, PGRP supports that it is actually possible to significantly restrict such attacks without being at the expense of user login convenience, and that the often believed trade-off between user convenience and login security with respect to password guessing being inevitable is not actually true.

8.5 Future Directions

While the focus of this thesis is on network scanning initiated by remote adversaries (i.e., remote to local), we believe that the proposed scan detection techniques apply to local scanners whose scanning targets are inside the monitored network. However, further experiments are necessary to evaluate the performance of these algorithms in such environments. For detecting local scanners that scan outside the monitored network, the challenge is in distinguishing between benign and malicious outbound failed connection attempts since the detection algorithm has less information about whether a targeted remote host offers (or did offer) a network service (on the targeted port).

IPv6 deployment (with 128-bit addresses relative to 32-bit addresses in IPv4) has been much slower than predicted. This could be the result of the introduced overhead of using IPv6 infrastructure (e.g., administration and cost) and the temporary solutions proposed to overcome the IPv4 small address domain space (e.g., NAT).

However, given that some networks support IPv6 protocol and that the pool of available IPv4 addresses is quickly running out (as the last top level (/8) block of free IPv4 addresses was assigned recently and that free IPv4 addresses are only available in already assigned blocks), there is a need to take into account IPv6 addresses in network scanning detection and login protocols (see [23]). Note, though, that it seems necessary that any host with IPv6 address should also have an IPv4 (e.g., through a NAT mechanism) to be able to communicate with hosts that have only IPv4 addresses, particularly since a significant number of hosts seems to remain with only IPv4 addresses for several years to come.

We believe that it is important to explore the impact of using IPv6 on performing network scanning with respect to the properties discussed in Chapter 4. Generally, given the 96-bit increase in IPv6 address space, finding responsive network services with random scanning seems infeasible in terms of both required resources and time. On the other hand, given the difficulty in identifying the number of remote hosts that share the same high-order 64 bits of an 128 bit IPv6 address, a single compromised host seems sufficient for a scanner to evade detection. This is because most ISP's end users are currently allocated a minimum of 2^{64} IPv6 addresses (i.e., the low-order 64 bits of the 128 bit IPv6 address) that can be all assigned to a single host, whereas the low-order 64 bits of the 128 bit IPv6 address can also be assigned to multiple machines in an enterprise subnet(s), and thus it seems unreasonable to flag the entire /64 network as a scanning source.

We believe that the required changes in network scanning detection algorithms (TRW, STRW, and LQS) must be studied to support IPv6 and the corresponding implications. To the best of our knowledge, there is no proposed scan detection algorithm that is designed or would work well with IPv6 addresses. For detection mechanisms based on remote hosts' connection failure ratio, the IPv6 support is likely to require more computational resources in terms of memory space to store the IPv6 addresses. If such extra memory space is not available for the detector, scanning detection accuracy is expected to significantly decrease. Similarly, an algorithm's immunity to attacks and scalability might change. Also, there are several tactics that adversaries could utilize to narrow down the vast IPv6 address space to smaller

subsets of a higher probability of finding addresses in use (e.g., see [15]).

It is also important to examine the effect of IPv6 on login protocols designed to address large-scale online dictionary attacks. IP blacklisting and whitelisting mechanisms become significantly harder to manage with IPv6 as it is difficult to identify whether a range of IPv6 addresses belongs to a single remote host or being part of a network subnet.

While we have experimented on several datasets from different network environments, we plan to further test the studied scan detection algorithms on other network traces, preferably from network environments different than those where we collected our datasets.

Bibliography

[1] Amazon Mechanical Turk. Accessed: June 2010. https://www.mturk.com/mturk/. 161

[2] Bro intrusion detection system. Accessed: March 2009, http://bro-ids.org/. 22, 36, 48, 77, 85, 99, 105, 113, 117, 118, 119, 132

[3] KDD cup data. http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.htmll. 128

[4] Tcpdpriv. Accessed: July 2010. http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html. 134

[5] C. Adams, G. Jourdan, J. Levac, and F. Prevost. Lightweight protection against brute force login attacks on web applications. In *Eighth Annual International Conference on Privacy Security and Trust (PST)*, pages 181 –188, aug 2010. 159

[6] M. Allman, V. Paxson, and J. Terrell. A brief history of scanning. In *Proc. the 7th ACM SIGCOMM Conference on Internet Measurement (IMC'07)*, pages 77–82, San Diego, California, USA, 2007. ACM. 2, 3, 78, 101, 146

[7] M. Alsaleh, D. Barrera, and P. van Oorschot. Improving security visualization with exposure map filtering. In *Proc. of the 24th Annual Computer Security Applications Conference (ACSAC'08)*, 2008. xii, 11, 40, 41

[8] M. Alsaleh, M. Mannan, and P. van Oorschot. Revisiting defenses against large-scale online password guessing attacks. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 99(PrePrints), 2011. 11, 155

[9] M. Alsaleh and P. C. van Oorschot. Revisiting network scanning detection using sequential hypothesis testing. In *Journal of Security and Communication Networks (to appear, 2012), John Wiley & Sons. Preliminary version as Technical Report TR-11-08, School of Computer Science, Carleton University, June 2011.* 11, 190

[10] M. Alsaleh and P. C. van Oorschot. Evaluation in the absence of absolute ground truth: Towards reliable evaluation methodology for scan detectors. In *Manuscript*, Jun 2011. 11, 150

[11] M. Alsaleh and P. C. van Oorschot. Network scan detection with LQS: A lightweight, quick and stateful algorithm. In *Proc. ASIACCS*, 2011. 10, 190

[12] M. Alsaleh, D. Whyte, and P. C. van Oorschot. A comparative evaluation of behaviour-based scanning detection algorithms based on the state of inbound connections. In *Manuscript in preparation*. 11

[13] S. Axelsson. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and System Security (TISSEC)*, 3(3):186–205, 2000. 145, 146

[14] R. Basu, R. K. Cunningham, and S. E. Webster. Detecting low-profile probes and novel denial-of-service attacks. In *Proc. the IEEE Workshop on Information Assurance and Security*, 2001. 25

[15] S. Bellovin, B. Cheswick, and A. Keromytis. Worm propagation strategies in an IPv6 internet. *LOGIN: The USENIX Magazine*, 31(1):70–76, 2006. 200

[16] S. M. Bellovin. A technique for counting natted hosts. In *ACM SIGCOMM Workshop on Internet measurment*, pages 267–272, New York, NY, USA, 2002. ACM. 165

[17] E. Bursztein, S. Bethard, J. C. Mitchell, D. Jurafsky, and C. Fabry. How good are humans at solving CAPTCHAs? A large scale evaluation. In *IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2010. 157, 178

[18] C. Muelder, K.-L. Ma, and T. Bartoletti. Interactive visualization for network and port scan detection. In *Proc. RAID*, 2005. 26

[19] M. Casado and M. J. Freedman. Peering through the shroud: The effect of edge opacity on IP-based client identification. In *4th USENIX Symposium on Networked Systems Design and Implementation (NDSS'07)*, 2007. 142, 158, 165

[20] CERT. Advanced scanning. CERT Incident Note IN-98.04 (Sept. 29 1998). http://www.cert.org/incident_notes/IN-98.04.html. 87

[21] CERT Network Situational Awareness Team (CERT NetSA). SiLK, the System for Internet-Level Knowledge. http://tools.netsa.cert.org/silk/index.html. Accessed: March 2009. 24

[22] S. Chiasson, P. C. van Oorschot, and R. Biddle. A usability study and critique of two password managers. In *USENIX Security Symposium*, pages 1–16, Vancouver, B.C., Canada, 2006. 173

[23] T. Chown. IPv6 implications for network scanning. http://www.ietf.org/rfc/rfc5157.txt. 4, 199

[24] G. J. Conti and K. Abdullah. Passive visual fingerprinting of network attack tools. In *Proc. Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC 2004)*. ACM, 2004. 26

[25] S. E. Coull, C. V. Wright, F. Monrose, M. P. Collins, and M. K. Reiter. Playing devil's advocate: Inferring sensitive information from anonymized network traces. In *NDSS*, 2007. 134

[26] M. de Vivo, E. Carrasco, G. Isern, and G. O. de Vivo. A review of port scanning techniques. *SIGCOMM Comput. Commun. Rev.*, 29:41–48, April 1999. 131

[27] Ethereal. Ethereal display filter reference. Accessed: Aug 2010. http://www.ethereal.com/docs/dfref/. 67

[28] V. Falletta and F. Ricciato. Detecting scanners: Empirical assessment on a 3G network. *International Journal of Network Security*, 9(2):143–155, September 2009. 23

[29] D. Florêncio, C. Herley, and B. Coskun. Do strong web passwords accomplish anything? In *USENIX workshop on Hot topics in security (HotSec'07)*, pages 1–6, Berkeley, CA, USA, 2007. 171

[30] S. Floyd and V. Paxson. Difficulties in simulating the internet. *IEEE/ACM Transactions on Networking*, 9:392–403, August 2001. 133

[31] K. Fu, E. Sit, K. Smith, and N. Feamster. Dos and don'ts of client authentication on the web. In *USENIX Security Symposium*, pages 251–268, Washington, DC, USA, 2001. 164

[32] M. Fullmer and S. Romig. The OSU Flow-tools package and Cisco Netflow logs. In *Proc. the 14th Systems Administration Conference (LISA'00)*, pages 291–303, New Orleans, LA, USA, 2000. USENIX Association. 22

[33] Fyodor, Phrack Magazine. The art of port scanning. Volume 7, Issue 51 September 01, 1997, article 11. http://www.phrack.org/issues.html?issue=51&id=11. 131

[34] C. Gates. Co-ordinated port scans: A model, a detector and an evaluation methodology. *PhD thesis, Dalhousie University, Canada*, 2006. 15, 128

[35] C. Gates. Coordinated scan detection. In *Proc. NDSS*, 2009. 15, 20, 32, 101

[36] C. Gates, J. J. McNutt, J. B. Kadane, and M. Kellner. Scan detection on very large networks using logistic regression modeling. In *Proc. the 11th IEEE Symposium on Computers and Communications (ISCC'06)*, 2006. 3, 20, 24, 132

[37] A. Graps. An introduction to wavelets. *Computing in Science and Engineering*, 2(2):50–61, 1995. 31

[38] P. Hansteen. Rickrolled? Get Ready for the Hail Mary Cloud! http://bsdly.blogspot.com/2009/11/rickrolled-get-ready-for-hail-mary.html. Accessed: Feb. 2010. 156

[39] Y. He and Z. Han. User authentication with provable security against online dictionary attacks. *Journal of Networks (JNW)*, 4(3):200–207, May 2009. 160

[40] L. T. Heberlein, G. V. Dias, K. N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. A network security monitor. *Proc. IEEE Symposium on Security and Privacy*, 1990. 21, 132

[41] B. V. Irwin and J.-P. van Riel. InetVis: a graphical aid for the detection and visualisation of network scans. In *Proc. the Workshop on Visualization for Computer Security (VizSEC'07)*, Sacramento, CA, USA, 2007. IEEE Computer Society. 27

[42] R. Jin and Z. Ghahramani. Learning with multiple labels. *Advances in Neural Information Processing Systems*, pages 921–928, 2003. 138, 139

[43] J. Jung. Real-time detection of malicious network activity using stochastic models. *PhD thesis, MIT*, 2006. 4, 48, 77, 118, 130

[44] J. Jung, R. A. Milito, and V. Paxson. On the adaptive real-time detection of fast-propagating network worms. In *Proc. DIMVA*, 2007. 20, 30, 95

[45] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proc. IEEE Symposium on Security and Privacy*, 2004. 3, 4, 7, 8, 14, 22, 32, 33, 36, 44, 45, 47, 54, 73, 85, 93, 110, 115, 119, 132, 141, 143, 146, 190, 193

[46] M. G. Kang, J. Caballero, and D. Song. Distributed evasive scan techniques and countermeasures. In *Proc. the Fourth GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'07)*, Lucerne, Switzerland, July 2007. 61, 62, 88, 96, 130

[47] T. Karagiannis, A. Broido, N. Brownlee, K. C. Claffy, and M. Faloutsos. Is p2p dying or just hiding? In *GLOBECOM Conference*, Nov 2004. 113

[48] N. Kato, H. Nitou, K. Ohta, G. Mansfield, and Y. Nemoto. A real-time IDS for large scale networks and its evaluations. *IEICE Transactions on Communication E82B(11)*, E82-B(11):1817–1825, November 1999. 62

[49] 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. 21, 132

[50] H. Kim, S. Kim, M. A. Kouritzin, and W. Sun. Detecting network portscans through anomaly detection. In *Proc. Signal Processing, Sensor Fusion, and Target Recognition XIII (SPIE'04)*, volume 5429, pages 254 – 263, 2004. 3, 21, 132

[51] E. Kirda, C. Kruegel, G. Vigna, and N. Jovanovic. Noxes: a client-side solution for mitigating cross-site scripting attacks. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 330–337. ACM, 2006. 165

[52] T. Kohno, A. Broido, and K. C. Claffy. Remote physical device fingerprinting. In *IEEE Symposium on Security and Privacy*, pages 211–225, Washington, DC, USA, 2005. IEEE Computer Society. 165

[53] B. Kosko. Bidirectional associative memories. *IEEE Transactions on Systems, Man and Cybernetics*, 18(1):49 –60, jan/feb 1988. 31

[54] M. Krzywinski. Port knocking from the inside out. *SysAdmin Magazine*, 12(6):12–17, 2003. 4

[55] K. Lakkaraju, W. Yurcik, and A. J. Lee. NVisionIP: netflow visualizations of system state for security situational awareness. In *Proc. the 2004 ACM workshop on Visualization and data mining for computer security (VizSEC'04)*, pages 65–72. ACM, 2004. 26

[56] L. Lam and S. Suen. Application of majority voting to pattern recognition: An analysis of its behavior and performance. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 27(5):553–568, 1997. 138

[57] S. Lau. The spinning cube of potential doom. *Communications of the ACM*, 47(6):25–26, 2004. 25

[58] C. Leckie and R. Kotagiri. A probabilistic approach to detecting network scans. In *Proc. NOMS*, 2002. 14, 21, 132

[59] Z. Li, A. Goyal, and Y. Chen. Honeynet-based botnet scan traffic analysis. In *Botnet Detection*, pages 25–44, 2008. 2, 101

[60] Z. Li, A. Goyal, Y. Chen, and V. Paxson. Automating analysis of large-scale botnet probing events. In *ASIACCS*, 2009. 2, 101

[61] R. Lippmann, J. Haines, D. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34(4):579–595, 2000. 128

[62] R. P. Lippmann, R. K. Cunningham, D. J. Fried, I. Graf, K. R. Kendall, S. E. Webster, and M. A. Zissman. Results of the DARPA 1998 offline intrusion detection evaluation. In *Proc. the Symposium on Recent Advances in Intrusion Detection (RAID'99)*, 1999. 128

[63] M. V. Mahoney and P. K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *Proc. the Sixth International Symposium on Recent Advances in Intrusion Detection (RAID'03)*, 2003. 128

[64] J. Mason, S. Small, F. Monrose, and G. MacManus. English shellcode. In *Proc. the 16th ACM conference on Computer and communications security*, 2009. 129

[65] J. McHugh. Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)*, 3:262–294, November 2000. 128

[66] J. McPherson, K.-L. Ma, P. Krystosk, T. Bartoletti, and M. Christensen. Portvis: a tool for port-based detection of security events. In *Proc. the 2004 ACM workshop on Visualization and data mining for computer security (VizSEC/DMSEC'04)*, pages 73–81. ACM, 2004. 26

[67] T. Moore and R. Clayton. Evil searching: Compromise and recompromise of internet hosts for phishing. In *Proc. FC*, 2009. 2

[68] M. Motoyama, K. Levchenko, C. Kanich, D. Mccoy, G. M. Voelker, and S. Savage. Re: CAPTCHAs understanding CAPTCHA-solving services in an economic context. In *USENIX Security Symposium*, Washington, DC, USA, Aug. 2010. 157

[69] C. Muelder, L. Chen, R. Thomason, K.-L. Ma, and T. Bartoletti. Intelligent classification and visualization of network scans. In *Proc. the Workshop on Visualization for Computer Security (VizSEC'07)*, October 2007. 31

[70] C. Muelder, K.-L. Ma, and T. Bartoletti. A visualization methodology for characterization of network scans. In *Proc. the Workshop on Visualization for Computer Security (VizSEC'05)*, pages 29–38, Washington, DC, USA, 2005. IEEE Computer Society. 31

[71] S. Murugesan. Understanding web 2.0. *IT professional*, 9(4):34–41, 2007. 29

[72] V. Nagaonkar. Detecting stealthy scans and scanning patterns using threshold random walk. *Master's thesis, Dalhousie University, Canada*, 2008. 23, 47, 52

[73] C. Namprempre and M. N. Dailey. Mitigating dictionary attacks with text-graphics character CAPTCHAs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E90-A(1):179–186, 2007. 165

[74] A. Narayanan and V. Shmatikov. Fast dictionary attacks on human-memorable passwords using time-space tradeoff. In *ACM Computer and Communications Security (CCS'05)*, pages 364–372, Alexandria, VA, USA, Nov. 2005. 156

[75] National Institute of Standards and Technology (NIST). hashbelt. Accessed: Sept. 2010. http://www.itl.nist.gov/div897/sqg/dads/HTML/hashbelt.html. 162

[76] 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 *Proc. LISA'00*, New Orleans, LA, USA, 2000. 22

[77] NetworkWorld.com. The biggest cloud on the planet is owned by ... the crooks. News article (Mar. 22, 2010). http://www.networkworld.com/community/node/58829. 156

[78] J. Nielsen. Stop password masking. Online article (June 23, 2009). http://www.useit.com/alertbox/passwords.html. 173

[79] NMAP. Accessed: Apr 2008, http://nmap.org. 2, 18, 62

[80] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 27–40. ACM, 2004. 22

[81] S. Panjwani, S. Tan, K. M. Jarrin, and M. Cukier. An experimental evaluation to determine if port scans are precursors to an attack. *Proc. Inter. Conference on Dependable Systems and Networks*, 2005. 2

[82] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proc. USENIX Security*, 1998. 27, 36

[83] V. Paxson, R. Pang, M. Allman, M. Bennett, J. Lee, and B. Tierney. LBNL/ICSI enterprise tracing project. http://www.icir.org/enterprise-tracing/download.html. Accessed: November 2008. 45

[84] B. Pinkas and T. Sander. Securing passwords against dictionary attacks. In *ACM conference on Computer and communications security (CCS'02)*, pages 161–170, Washington, DC, USA, Nov. 2002. xiii, 7, 9, 157, 158, 159, 161, 166, 168, 169, 175, 179, 180, 181, 182, 183, 184, 186, 194, 198

[85] T. Ptacek, T. Newsham, and H. J. Simpson. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks, Inc., January 1998. 130

[86] D. Ramsbrock, R. Berthier, and M. Cukier. Profiling attacker behavior following SSH compromises. In *IEEE/IFIP Dependable Systems and Networks (DSN'07)*, pages 119–124, Edinburgh, UK, June 2007. 156

[87] H. Ringberg, M. Roughan, and J. Rexford. The need for simulation in evaluating anomaly detectors. *SIGCOMM Comput. Commun. Rev.*, 38:55–59, January 2008. 133

[88] H. Ringberg, A. Soule, and J. Rexford. WebClass: adding rigor to manual labeling of traffic anomalies. *SIGCOMM Comput. Commun. Rev.*, 38:35–38, January 2008. 133

[89] S. Robertson, E. V. Siegel, M. Miller, and S. J. Stolfo. Surveillance detection in high bandwidth environments. In *Proc. the DARPA DISCEX III Conference*, pages 130–139, April 2003. 3, 21, 31

[90] D. Roelker, M. Norton, and J. Hewlett. sfPortscan, September 2004. http://cvs.snort.org /viewcvs.cgi/snort/doc/README. sfportscan?rev=1.6. 8, 22, 85, 132

[91] M. Roesch. Snort: lightweight intrusion detection for networks. In *Proc. LISA*, 1999. 22, 27, 132

[92] S. Sanfilippo. Bugtraq: New TCP scan method. http://seclists.org/bugtraq/1998/Dec/79. 18

[93] SANS.org. Important information: Distributed SSH brute force attacks. SANS Internet Storm Center handler's diary (June 18, 2010). http://isc.sans.edu/diary.html?storyid=9034. 156

[94] SANS.org. The top cyber security risks. Online article (Sept. 2009). http://www.sans.org/top-cyber-security-risks/. 5, 156, 170

[95] S. E. Schechter, J. Jung, and A. W. Berger. Fast detection of scanning worm infections. In *Proc. RAID*, 2004. 2, 20, 28, 30, 62, 95

[96] V. Sheng, F. Provost, and P. Ipeirotis. Get another label? Improving data quality and data mining using multiple, noisy labelers. In *Proc. conference on Knowledge discovery and data mining*, 2008. 139

[97] G. J. Simon and H. Xiong. Scan detection: A data mining approach. In *Sixth SIAM International Conference on Data Mining*, 2006. 3, 23, 25, 83, 143, 146, 147

[98] Solar Designer, Phrack Magazine. Designing and attacking port scan detection tools. Volume 8, Issue 53 July 8, 1998, article 13. http://www.phrack.org/issues.html?issue=53&id=13article. 132

[99] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *IEEE Symposium on Security and Privacy*, may 2010. 127, 133

[100] A. Sridharan and T. Ye. Implementing real time port scan detection for the IP backbone. Sprint ATL Research Report RR07-ATL-020399, 2007. 24

[101] A. Sridharan and T. Ye. Tracking port scanners on the IP backbone. In *Proc. LSAD*, 2007. 3, 20, 24

[102] A. Sridharan, T. Ye, and S. Bhattacharyya. Connectionless port scan detection on the backbone. *Performance, Computing, and Communications Conference (IPCCC'06)*, page 76, 2006. 3, 24

[103] S. Stafford and J. Li. Behavior-based worm detectors compared. In *Proc. RAID*, 2010. 9, 105, 195

[104] S. Staniford. Containment of scanning worms in enterprise networks. *Journal of Computer Security*, November 2003. 20, 28

[105] 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. 3, 15, 21, 130, 132

[106] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the Internet in your spare time. In *Proc. the 11th USENIX Security Symposium*, pages 149–167, 2002. 28

[107] 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 *Proc. the 19th National Information Systems Security Conference*, pages 361–370, 1996. 25

[108] K. Stockinger, E. W. Bethel, S. Campbell, E. Dart, and K. Wu. Detecting distributed scans using high-performance query-driven visualization. In *Proc. the ACM/IEEE Conference on Supercomputing*, 2006. 32

[109] C. Stoll. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. Doubleday, 1989. 186

[110] W. W. Streilein, R. K. Cunningham, and S. E. Webster. Improved detection of low-profile probe and DoS attacks. In *Proc. the Workshop on Statistical and Machine Learning Techniques in Computer Intrusion Detection*, 2001. 25, 30

[111] F. Thabtah, P. Cowling, and Y. Peng. Multiple labels associative classification. *Knowledge and Information Systems*, 9(1):109–129, 2006. 139

[112] TheRegister.co.uk. Botnet pierces Microsoft Live through audio captchas. News article (Mar. 22, 2010). http://www.theregister.co.uk/2010/03/22/microsoft_live_captcha_bypass/. 157, 170

[113] P. C. van Oorschot and S. Stubblebine. On countering online dictionary attacks with login histories and humans-in-the-loop. *ACM Transactions on Information and System Security (TISSEC)*, 9(3):235–258, 2006. xiii, 7, 9, 157, 158, 160, 166, 168, 169, 173, 174, 175, 179, 180, 181, 185, 194, 198

[114] J.-P. van Riel and B. Irwin. InetVis, a visual tool for network telescope traffic analysis. In *Proc. the 4th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH'06)*, pages 85–89. ACM, 2006. 27

[115] C. J. van Rijsbergen. *Information Retrieval*. 2nd Edition, London: Butterworths, 1979. http://www.dcs.gla.ac.uk/Keith/Preface.html. 145

[116] G. Vigna. Network intrusion detection: dead or alive? In *Proc. the 26th Annual Computer Security Applications Conference (ACSAC'10)*, 2010. 128

[117] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *Eurocrypt*, pages 294–311, Warsaw, Poland, May 2003. 156

[118] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms, revisited. *Malware Detection*, 27:113–145, 2007. Springer-Verlag Advances in Information Security. 2, 20, 23, 29, 47, 52

[119] M. Weir, S. Aggarwal, M. Collins, and H. Stern. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 162–175. ACM, 2010. 156

[120] D. Whyte. Network scanning detection strategies for enterprise networks. *PhD thesis, Carleton University*, 2008. 38, 45, 65

[121] D. Whyte, E. Kranakis, and P. C. van Oorschot. DNS-based detection of scanning worms in an enterprise network. In *Proc. the 12th Annual Network and Distributed System Security Symposium (NDSS'05)*, pages 181–195, 2005. 16, 29

[122] D. Whyte, P. C. van Oorschot, and E. Kranakis. Detecting intra-enterprise scanning worms based on address resolution. In *Proc. Annual Computer Security Applications Conference (ACSAC'05)*, pages 371–380. IEEE Computer Society, 2005. 29

[123] D. Whyte, P. C. van Oorschot, and E. Kranakis. Exposure maps: removing reliance on attribution during scan detection. In *Proc. the 1st USENIX Workshop on Hot Topics in Security (HOTSEC'06)*, Vancouver, B.C., Canada, 2006. USENIX Association. 7, 38, 44, 45, 190, 193

[124] D. Whyte, P. C. van Oorschot, and E. Kranakis. Tracking darkports for network defense. In *Proc. ACSAC*, 2007. 4, 10, 87, 106

[125] C. Wong, S. Bielski, A. Studer, and C. Wang. Empirical analysis of rate limiting mechanisms. In *Proc. the 8th International Symposium on Recent Advances in Intrusion Detection*, 2005. 28

[126] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber. How dynamic are IP addresses? *SIGCOMM Comput. Commun. Rev.*, 37(4):301–312, 2007. 168

[127] J. Yan and A. S. E. Ahmad. A low-cost attack on a Microsoft CAPTCHA. In *ACM Computer and Communications Security (CCS'08)*, pages 543–554, Alexandria, VA, USA, Oct. 2008. 157, 170

[128] J. Yan and A. S. E. Ahmad. Usability of CAPTCHAs or usability issues in CAPTCHA design. In *Symposium on Usable Privacy and Security (SOUPS'08)*, pages 44–52, Pittsburgh, PA, USA, July 2008. 157

[129] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: global characteristics and prevalence. *SIGMETRICS Perform. Eval. Rev.*, 31(1):138–147, 2003. 20

[130] V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: global characteristics and prevalence. In *Proc. the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*, pages 138–147, June 2003. 31

[131] Y. Zhang and B. Fang. A novel approach to scan detection on the backbone. In *Sixth International Conference on Information Technology: New Generations (ITNG'09)*, April 2009. 132

Appendix A

Implementation of LQS in Bro policy

```
# the current algorithm supports TCP only
module LQS;
export  {}

global k = 2;   # as in the pseudo-code in the LQS algorithm

type info2: record {
        Ports: set[port];
};

type info: record {
       Contacted: table[addr] of info2;
       Count: double &default = 0.0;
};

global OPS: set[addr, port] &write_expire = 7 day;
global CPS: set[addr, port] &write_expire = 1 day;
global FC: table[addr] of info &write_expire = 2 day;
global SC: set[addr];
global CR: set[addr, addr] &write_expire = 1 hr;
global OPS_count = 0;
global CPS_count = 0;
global hour_count = 0;
global day_count = 0;

event partial_connection(c: connection)
{
  local dst_host = c$id$resp_h;
  local dst_port = c$id$resp_p;
  local orig_host = c$id$orig_h;

  if (get_port_transport_proto(c$id$orig_p) == tcp)  {
    if ([orig_host] in FC)
        FC[orig_host]$Count = FC[orig_host]$Count - 1;
    else if ([dst_host] in FC)
        FC[dst_host]$Count = FC[dst_host]$Count - 1;
  }
}
```

```
# This event is when a new session is just started. For TCP, this is when a first SYN
    packet is seen
event new_connection(c: connection)
{
  local orig_host = c$id$orig_h;
  local dst_port = c$id$resp_p;
  local dst_host = c$id$resp_h;
  local os = c$orig$state;
  local temp1: info;
  local temp2: info2;

  # destination is local address, TCP only, source is not local, and connection not
      in the OPS or CPS tables
  if ( is_local_addr(dst_host) && (get_port_transport_proto(c$id$orig_p) == tcp) && !
      is_local_addr(orig_host) && [dst_host, dst_port] !in OPS && [dst_host, dst_port
      ] !in CPS && [dst_host, orig_host] !in CR)  {

    if ([orig_host] !in FC)  {
      FC[orig_host] = temp1;
    }

    if (FC[orig_host]$Count < k)  {
        if ([dst_host] !in FC[orig_host]$Contacted)
        {
          FC[orig_host]$Contacted[dst_host] = temp2;
          add FC[orig_host]$Contacted[dst_host]$Ports[dst_port];
          FC[orig_host]$Count = FC[orig_host]$Count + 1;
        }
        else if ([dst_port] !in FC[orig_host]$Contacted[dst_host]$Ports)
        {
          add FC[orig_host]$Contacted[dst_host]$Ports[dst_port];
          FC[orig_host]$Count = FC[orig_host]$Count + 0.25;
        }
        if (FC[orig_host]$Count >= k)
        {
          add SC[orig_host];
        }
    }
  }
  else if ( (is_local_addr(orig_host)) && (!is_local_addr(dst_host)) && (dst_host !in
      SC) && !(os == TCP_RESET) )  {
        # whitelist this remote for a while
        add CR[orig_host, dst_host];
  }
}

# This event is when a TCP session is complete (i.e., receiving a SYN-ACK packet
    after sendinga SYN packet)
```

```
event connection_established(c: connection)
{
  local dst_host = c$id$resp_h;
  local dst_port = c$id$resp_p;
  local orig_host = c$id$orig_h;

  # TCP only and remot to local dest
  if ( (get_port_transport_proto(c$id$orig_p) == tcp) && (is_local_addr(dst_host)) &&
       !is_local_addr(orig_host) )  {
    add OPS[dst_host, dst_port]; # just to refresh the write_expire
    if ( [dst_host, dst_port] in CPS )  {
      add OPS[dst_host, dst_port];
      delete CPS[dst_host, dst_port];
    }
    if ([orig_host] in FC)
      if ([dst_host] in FC[orig_host]$Contacted)
        if  ([dst_port] in FC[orig_host]$Contacted[dst_host]$Ports)  {
          delete FC[orig_host]$Contacted[dst_host]$Ports[dst_port];
          if (length(FC[orig_host]$Contacted[dst_host]$Ports) > 0)
            FC[orig_host]$Count = FC[orig_host]$Count - 0.25;

          # all connections to this local host are successful, so local host should
              be deleted
          else  {
            FC[orig_host]$Count = FC[orig_host]$Count - 1;
            delete FC[orig_host]$Contacted[dst_host];
          }
          if (FC[orig_host]$Count == 0)
            delete FC[orig_host];
        }
  }
}

event connection_rejected(c: connection)
{
  local dst_host = c$id$resp_h;
  local dst_port = c$id$resp_p;
  local orig_host = c$id$orig_h;

  if ( (get_port_transport_proto(c$id$orig_p) == tcp) && (is_local_addr(dst_host)) &&
       ([dst_host, dst_port] in OPS) )  {
    add CPS[dst_host, dst_port];
    delete OPS[dst_host, dst_port];
  }
}
```

Appendix B

Implementation of STRW in Bro policy

```
module STRW;

export {
  # Activate STRW if T.
  global use_STRW_algorithm = T &redef;

  # Tell STRW not to flag a friendly remote
  global do_not_flag_friendly_remotes = T &redef;

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

  # The following correspond to P_D and P_F in the STRW paper, i.e., the
  # desired detection and false positive probabilities.
  global target_detection_prob = 0.99 &redef;
  global target_false_positive_prob = 0.01 &redef; # was 0.01

  # Given a legitimate remote, the probability that its connection
  # attempt will succeed.
  global theta_zero = 0.8 &redef; # was 0.8

  # Given a scanner, the probability that its connection attempt
  # will succeed.
  global theta_one  = 0.2 &redef;
  global external_count = 0;
  global internal_count = 0;
  global reverse_count = 0;

  # These variables the user usually won't alter, except they
  # might want to adjust the expiration times, which is why
  # they're exported here.
  global scan_sources: table[addr] of time; # &write_expire = 1 hr;
  global benign_sources: table[addr] of time; # &write_expire = 1 hr;

  global failed_locals: set[addr, addr] &write_expire = 30 mins;
  global successful_locals: set[addr, addr] &write_expire = 30 mins;
  global lambda: table[addr] of double &default = 1.0 &write_expire = 1 day;
```

```
  # Function called to perform STRW analysis.
  global check_STRW_scan: function(c: connection, state: string, reverse: bool,
      connection_established: bool): bool;

  # keeping track of open ports in the local network
  global services: set[addr, port] &write_expire = 3 days;
}

# Set of remote hosts that have been successfully accessed by local hosts.
global friendly_remotes: set[addr] &read_expire = 30 mins;

# Approximate solutions for upper and lower thresholds.
global eta_zero: double;
global eta_one: double;

function check_STRW_scan(c: connection, state: string, reverse: bool,
    connection_established: bool): bool
{
  local id = c$id;
  local service = "ftp-data" in c$service ? 20/tcp:(reverse?id$orig_p:id$resp_p);
  local orig = reverse ? id$resp_h : id$orig_h;
  local resp = reverse ? id$orig_h : id$resp_h;
  local outbound = is_local_addr(orig);
  local p = reverse ? id$orig_p : id$resp_p;
  local ts = c$start_time;
  local dst_port = reverse ? id$orig_p : id$resp_p;

  # packet coming from an external source and going to an external source (to solve
      the problem of unknown locals)
  if ( (!is_local_addr(orig))&&(!is_local_addr(resp)) )
    return F;

  # Mark a remote as friendly if it is successfully accessed by
  # a local with protocols other than triggered_outbound_services.
  # XXX There is an ambiguity to determine who initiated a
  # connection when the status is "OTH".
  if ( outbound ) {
    if ( resp !in scan_sources && service !in triggered_outbound_services && state !=
        "OTH" )
      add friendly_remotes[resp];
    return F;
  }

  # Adding to the network services table
  if ((connection_established)&&(is_local_addr(resp))) {
    add services[resp, dst_port]; # if already exists in services then it will
        refresh the write_expire
  }
```

```
if ( orig in scan_sources ) {
  return T;
}

if ( orig in benign_sources ) {
  return F;
}

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

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

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

# Changing the decision oracle
if ( (established) && ( [orig, resp] !in successful_locals ) ) {
  flag = -1;
  add successful_locals[orig, resp];
}
else if ( ([resp, dst_port] !in services) ) {
       if ( [orig, resp] !in failed_locals ) {
         flag = 1;
         add failed_locals[orig, resp];
       }
 }

if ( flag == 0 )
  return F;

local ratio = 1.0;

# Update the corresponding likelihood ratio of orig.
if ( theta_zero <= 0 || theta_zero >= 1 || theta_one <= 0 || theta_one >= 1 ||
    theta_one >= theta_zero ) {
  # Error: theta_zero should be between 0 and 1.
  alarm "bad theta_zero or theta_one in check_STRW_scan";
  use_STRW_algorithm = F;
  return F;
}
```

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

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

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

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

  if ( updated_lambda > eta_one ) {
    scan_sources[orig] = ts;
    return T;
  }

  if ( updated_lambda < eta_zero ) {
    benign_sources[orig] = ts;
  }

  return F;
}


# *** connection events ***

function conn_state(c: connection, trans: transport_proto): string
{
  local os = c$orig$state;
  local rs = c$resp$state;
  local o_inactive = os == TCP_INACTIVE || os == TCP_PARTIAL;
  local r_inactive = rs == TCP_INACTIVE || rs == TCP_PARTIAL;

  if ( trans == tcp ) {
    if ( rs == TCP_RESET ) {
      if ( os == TCP_SYN_SENT || os == TCP_SYN_ACK_SENT || (os == TCP_RESET &&
          c$orig$size == 0 && c$resp$size == 0) )
        return "REJ";
      else if ( o_inactive )
        return "RSTRH";
      else
        return "RSTR";
```

```
    }
    else if ( os == TCP_RESET )
      return r_inactive ? "RSTOS0" : "RSTO";
    else if ( rs == TCP_CLOSED && os == TCP_CLOSED )
      return "SF";
    else if ( os == TCP_CLOSED )
      return r_inactive ? "SH" : "S2";
    else if ( rs == TCP_CLOSED )
      return o_inactive ? "SHR" : "S3";
    else if ( os == TCP_SYN_SENT && rs == TCP_INACTIVE )
      return "S0";
    else if ( os == TCP_ESTABLISHED && rs == TCP_ESTABLISHED )
      return "S1";
    else
      return "OTH";
  }
  else if ( trans == udp )  {
    if ( os == UDP_ACTIVE )
      return rs == UDP_ACTIVE ? "SF" : "S0";
    else
      return rs == UDP_ACTIVE ? "SHR" : "OTH";
  }
  else
    return "OTH";
}


event connection_established(c: connection)
{
  local is_reverse_scan = (c$orig$state == TCP_INACTIVE);
  local trans = get_port_transport_proto(c$id$orig_p);

  if (is_reverse_scan)
    ++reverse_count;

  if ( trans == tcp && STRW::use_STRW_algorithm )
    STRW::check_STRW_scan(c, conn_state(c, trans), F, T);
}


event connection_attempt(c: connection)
{
  local trans = get_port_transport_proto(c$id$orig_p);

  if ( trans == tcp && STRW::use_STRW_algorithm )
    STRW::check_STRW_scan(c, conn_state(c, trans), F, F);
}
```

```
event connection_rejected(c: connection)
{
  local is_reverse_scan = c$orig$state == TCP_RESET;
  local trans = get_port_transport_proto(c$id$orig_p);

  if (is_reverse_scan)
    ++reverse_count;

  if ( trans == tcp && STRW::use_STRW_algorithm )
    STRW::check_STRW_scan(c, conn_state(c, trans), is_reverse_scan , F);
}
```