NETWORK SCANNING DETECTION STRATEGIES FOR ENTERPRISE NETWORKS

by

David Whyte

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of

DOCTOR OF PHILOSOPHY

School of Computer Science

at

CARLETON UNIVERSITY

Ottawa, Ontario
September, 2008

Table of Contents

List of Tables

List of Figures

Abstract

The Internet is saturated with nonproductive network traffic that includes a variety of reconnaissance activities to identify vulnerable systems. Individual systems exhibit anomalous behavior in their interactions with physical and logical interconnections that define the enterprise network when they are scanning or are the target of a scan. We take advantage of this observation through the development of a suite of network scanning detection techniques to detect internal (intra-enterprise) scanning using the address resolution protocols (i.e. Domain Name System (DNS), Address Resolution Protocol (ARP)), and external (inter-enterprise) scanning using *darkports* – the unused ports on active systems, which we identify during the construction of *exposure maps*.

Specifically, to detect intra-enterprise network scanning activity, we note that scanning systems exhibit anomalous behavior when using the address resolution protocols. These techniques offer the possibility to identify local scanning systems within an enterprise network after the observation of only a few scanning attempts with a low false positive and negative rate. To detect external scanning activity directed at a network we make use of the concept of exposure maps that are identified by passively characterizing the connectivity behavior of internal hosts in a network as they respond to both legitimate connection attempts and scanning attempts. The exposure maps technique enables: (1) active response options to be safely focused exclusively on those systems that directly threaten the network, (2) the ability to rapidly characterize and group hosts in a network into different exposure profiles based on the services they offer, and (3) the ability to perform a Reconnaissance Activity Assessment (RAA) that determines what specific information was returned to an adversary as a result of a directed scanning campaign.

In a direct side-by-side comparison with the Threshold Random Walk (TRW) scanning detection technique of Jung (2006, MIT Ph.D., thesis) exposure maps offered an equivalent scanning detection capability while arguably being lightweight, and offering additional functionality. This dissertation describes the design, implementation and evaluation of fully functional prototypes to detect internal and external scanning activity at an enterprise network.

Acknowledgements

I would like to express my sincere gratitude to my advisor Professor Evangelos Kranakis. Your expert knowledge, unfailing encouragement and personal guidance has been a source of inspiration to me for a number of years. You gave me the confidence to pursue this degree and allowed me to follow and realize my dreams.

I am deeply grateful to my advisor Professor Paul C. van Oorschot for the detailed, insightful, and constructive comments he provided not only for this thesis but for all my research and publications. I cannot overstate how important your unfailing support, personal guidance, and excellent advice has benefited me. I especially thank you for the many hours and late nights you spent helping me improve this thesis. You have had a remarkable influence on my academic research.

I would also like to take this opportunity to thank the members of my thesis committee that consisted of Professor Tet Yeap, Professor Anil Somayaji, Professor Ashraf Matrawy and Professor John McHugh. A special mention is warranted for both Professor Somayaji and Professor McHugh whose helpful comments and keen observations have certainly improved the quality of this work.

I extend a special thanks to all the members of the Digital Security Group at the Carleton Computer Security Lab. The lab has proved not only to be a great facility to conduct research but also a wonderful environment where top notch security researchers can both challenge and inspire one another. Thanks to Glenn Wurster and Mohammad Mannan for your continual support and encouragement over the last few years.

I am indebted to my many professional colleagues for whom I have great regard. To all of my colleagues, I appreciate your advice, comments, and support. I extend a special thanks to Griffith Ince, Carl Reid and Deborah Abbott who made it possible for me to devote full-time study at critical points in my research. I would not have been able to complete this degree without these times of focused research. For that, I offer you all my heartfelt thanks and a debt of gratitude. To Donald MacLeod and Joe MacGillivray, your expert technical advice both professionally and academically has been of immeasurably benefit.

Lastly, and most importantly, I wish to dedicate this thesis to my family. To my parents, Donald and Sharon, thank you for instilling in me the importance of education and for always believing in my abilities. To my wife Michelle, this would not have been possible without your love, support, and encouragement. As I completed this work, you spent many nights and weekends alone taking care of our children. In this, as in all things, you enable me to realize my potential. To my children Carter and Kylie, let this serve as proof that if you work hard you too can realize your dreams. Always remember, I will be there to help you along the way.

Chapter 1

Introduction

To safeguard an enterprise network, it is desirable that all forms of external and internal scanning activity be detected to identify information leakage to an adversary in an attempt to prevent follow-on attacks. To accomplish this goal, this dissertation presents network scanning detection strategies that: (1) are sufficiently accurate and efficient to support automated countermeasures (e.g. containment by ingress/egress blocking, connection throttling), (2) can detect both new (i.e. zero-day) and sophisticated forms of scanning activity, (3) provide a mechanism to determine what type of information about the network has been obtained by an adversary as a result of network scanning, and (4) are suitable to detect inter and intra-enterprise network scanning activity. The remainder of this chapter contains sections on the motivations for this work, an attempt to broadly classify our network detection strategies based on their underlying detection principles, a summary of the contributions of this dissertation, and finally an outline for subsequent chapters.

1.1 Motivation

Networks are under constant bombardment from a variety of *unproductive* network activity that includes probes from compromised systems (e.g. worms, auto-rooters [66]), misconfigured systems, Internet cartographers, and backscatter traffic [40]. Yegneswaren et al. [79] estimated that there were, already in the 2002 timeframe, 25 billion global intrusion attempts per day and this activity continues to increase. Panjwani et al. [44] estimated in a 2005 paper that 50% of attacks against systems are preceded by some form of network scanning activity. Network reconnaissance or scanning is the first stage of an intrusion attempt that enables an adversary to remotely locate, target, and exploit vulnerable systems. Network scanning activity directed at an enterprise network can occur from systems within (i.e. a host inside the administrative domain of the network scanning local subnets) or external to the network.

Automated tools (e.g. auto-rooters) methodically probe large blocks of Internet address space seeking vulnerable systems for recruitment into botnets [20, 51, 15, 47, 5]. Large numbers of worm-infected systems randomly scan the Internet searching for susceptible systems to exploit. Over the past few years, and perhaps best characterized by the the period 2001 – 2003 (with Code Red, Nimda, Slammer, and Blaster), worm outbreaks of very large size and severity have rapidly spread across vulnerable systems destabilizing the Internet. In fact, there have been worm outbreaks that have been able to scan and infect 90% of all the vulnerable host on the Internet in less than 10 minutes [39].

Perhaps most worrisome for a network operator is when a determined adversary directs specific scanning activity solely against their network searching for weaknesses that provide an entry vector. This type of reconnaissance is typically precise, deliberate, and focused. In contrast to the indiscriminate scanning activity typically associated with autorooter and worm propagation activities, skilled adversaries will go to considerable lengths to mask their activities.

Almost all current scanning detection algorithms correlate scanning activity based on the perceived last-hop origin of the scans; we call these *attribution-based* detection schemes. However, there are situations (i.e. in cases of remote scanning of an enterprise network) where determining *true* attribution (e.g. the actual scan controller, where this differs) is not possible. Furthermore, in some cases the use of attribution-based detection schemes is entirely ineffective as the scans may either be so slow or so broadly distributed that they exhaust the finite computational state of scanning detection systems or fail to exceed some predefined alert threshold. Although a network operator may be interested in knowing what type and amount of scanning activity is occurring, this is largely irrelevant if the proper security countermeasures are in place and relevant software patches are available and up-to-date. However, the situation is different if any of the scans are a more likely precursor to a successful attack. Current scanning detection techniques do not take advantage of this observation. Our view is that against a growing array of scanning strategies from remote hosts, attribution is becoming a quixotic approach to scan detection that overlooks an often critically important question that we suggest should be a much higher focus of scanning detection, namely, "What is the adversary looking for and, are we vulnerable with respect to that resource?"

Our thesis is that *network scanning detection techniques can both leverage and benefit from local knowledge obtained from considering an enterprise network's topology and the perceived origin (i.e. internal or external to the network) of the scanning activity.* Specifically, systems exhibit anomalous behavior in their interactions with physical and logical interconnections that define the enterprise network when they are scanning or are the target of a scan that can provide indications of scanning activity. The research presented in this thesis pursues the following 3 hypotheses.

Hypothesis 1: to better defend the enterprise network, it is possible to design and deploy a suite of practical scanning detection techniques, that improve upon existing approaches, with acceptably low false postive/negative rates, and that are responsive within a very small number of scans (e.g., 1 to 3 scans - as low as a single scanning attempt).

Related to hypothesis 1, this thesis explores 3 new techniques to detect the differing topological vantages of attack (i.e. remote to local, local to remote, and local to local) within an enterprise network.

Hypothesis 2: it is possible to make novel use of address resolution protocols to detect malicious network activity, including some zero-day worms.

Hypothesis 3: it is possible to devise a highly efficient scan detection technique which does not rely on who is doing the scanning, but rather on what service (or in general, what resource) is being scanned for. Here, as one example of efficiency, system state (in terms of main memory consumed) need not increase linearly with bursts in external scanning activity.

1.2 Network-based Intrusion Detection Schemes

Network scanning activities are typically undertaken to locate and identify active hosts in the network and the associated services they offer (i.e. designated by open ports). A variety of complex heuristics have been successfully developed to detect scanning activity including the observation of connection failures [29, 58], statistical measures [31, 64], abnormal network behaviors [71, 76, 18], and connections to network darkspace (i.e. the unused IP addresses within a network) [19, 36]. Those network scanning detection algorithms that identify remote host (i.e. hosts external to the local network) scanning activity can be considered attribution-based as they focus largely on observing and classifying external network behavior (i.e. incoming network connection attempts) to detect scanning systems and therefore experience the limitations described in the previous section.

Network scanning detection techniques are implemented in a number of network-based intrusion detection systems. A number of definitions exist that attempt to define network-based intrusion detection systems based on a characterization of their underlying detection principles, however, they can be broadly classified into three types (adapted from [59]): (1) misuse detection, (2) anomaly detection, and (3) specification-based detection.

- **Misuse detection** is implemented by comparing collected network data to known malicious intrusion signatures (e.g. patterns). Misuse detection systems typically do not require any training periods and are fairly simple to use and deploy. They are limited in that they are only able to detect malicious activity based on prior knowledge (e.g. some unique discernible pattern of the attack is known) and therefore new or novel attacks may not be detected.

- **Anomaly detection** is implemented by comparing the behavior of the network or systems that comprise it with a model of expected behavior. The model of normal behavior is typically constructed through the use of a training period or datasets and the use of machine learning techniques. In general, an important consideration for any technique that requires a training period is that any existing malicious activity may become part of the baseline. Techniques based on anomaly detection have the potential benefit of being able to detect new or novel attacks but they usually involve a time intensive endeavor to keep updated and can be complex to use. For instance, additional analysis will be required to identify both the type and severity of any detected attack.

- **Specification-based detection** is implemented by developing specifications of legitimate system behavior for comparison against observed system behavior. Although this technique is similar to anomaly-based detection, it does not rely on machine learning techniques but rather manually developed specifications of accepted system behavior. In practice, specification-based detection techniques avoid the typically high false positive rate experienced by anomaly-based techniques but at the expense of a significant amount of time to develop proper and comprehensive specifications.

The network scanning detection strategies developed in this thesis can be categorized as anomaly-based detection techniques. Specifically, our techniques rely on the passive observation of internal network hosts as they interact with hosts both within and external to the local enterprise network.

1.3 Contributions

The overall contributions of this dissertation can be summarized as follows.

1. **Internal Network Scanning Detection Techniques**: we present two new techniques to detect scanning systems within a local network based on the anomalous behaviors they exhibit when using the address resolution protocols (i.e. Address Resolution Protocol (ARP) [48], and Domain Name Service (DNS) [37]). Based on our evaluation, using datasets obtained from a small university network, these techniques offer a significant improvement over existing scanning detection techniques. Specifically, in certain network environments, these techniques can identify scanning hosts within an enterprise network after the observation of only a single scanning attempt with a low false positive and negative rate. Preliminary results appear in two published papers [71, 72].

   (a) **ARP-based detection**: this behavioral signature is based on the observation that a scanning host targeting systems within its own network exhibits anomalous behavior distinct from normal ARP activity; specifically, scanning hosts exhibit discernible behavioral changes in the amount and pattern of ARP request activity of the scanning host, because a scanning host targeting local network hosts triggers the broadcast of anomalous ARP requests.

   (b) **DNS-based detection**: we exploit the observation that the vast majority of publicly available services are accessed through the use of DNS as this protocol provides the mapping between numeric IP addresses and the corresponding alphanumeric names. Many fast scanning worms and automated scanning tools use a pseudo random number generator (PRNG) to generate 32-bit random numbers that correspond to an IPv4 address. The use of a numeric IP address, instead of the qualified domain name of the system, obviates the need for a DNS query. If we do not observe DNS activity before a new connection is initiated, we consider this connection anomalous and potential scanning activity.

2. **External Network Scanning Detection Strategies**: we present a technique that detects network scanning activity (both UDP and TCP scanning) directed against a local network from remote hosts. We exploit, for defensive purposes, the concept of *exposure maps* and *darkports*. Exposure maps are tables listing the open ports (ports actively providing service) on active systems. Darkports refer to the unused ports on active systems.

   (a) **Exposure maps**: are built by passively observing and characterizing the connectivity behavior of internal hosts in a network as they respond to both legitimate connection attempts and scanning attempts. Exposure maps differ from current scanning detection techniques as they rely on identifying the services offered by the network instead of tracking external connection events. The result is a scanning detection technique in which the utilized system detection state does not grow in proportion to the amount and fluctuation of external network traffic, but rather increases only with the number of services offered by the network, regardless of the size of the network and the external network activity. Preliminary results of this technique have been published [73, 75]. Unlike most attribution-based scanning detection techniques (that detect remote scanning systems), the scanning detection approach does not rely on identification of the scanning source to detect scans against a network. Thus, it can detect certain classes of sophisticated scanning techniques that make determining the root cause of the scanning activity impractical. However, this approach does not preclude the use of some form of attribution *post* scan detection. Scanning worm propagation and auto-rooters are two prevalent examples of scanning activities where immediately denying the scanning source access to the network is both relevant and important.

(b) **Comparison to Threshold Random Walk (TRW)**: TRW is a scanning detection technique [29, 28] and has been positioned as the *gold standard* in existing scanning detection algorithms [22]. We performed a limited comparison (i.e. only a few selected values of TRW parameters were chosen in our comparison; a more complete analysis would explore a full range of values) with TRW and a modified TRW technique we developed through augmentation with the exposure maps technique, using three different network datasets. Our results show that exposure maps offers an equivalent scanning detection capability while offering additional functionality. Namely, exposure maps can be used to detect sophisticated scanning activity, enable fine-grained automated defense against automated malware attacks, provide a mechanism to identify the network information gained by a successful scanning system, and detect real-time changes in a network that may indicate a successful compromise. As well, the modified TRW technique offers the possibility of better performance than TRW in terms of faster scan detection as the use of the NEM obviates the need to wait in the determination if a connection attempt will succeed or fail.

(c) **Use for sophisticated scan detection**: unlike existing scanning detection schemes that rely on the correlation of multiple network *events* (e.g. number of failed or successful connection attempts observed) to detect and report scanning activity, the exposure maps technique generates and records individual connection attempts to darkports (i.e. darkport connection attempts – DCAs) that can be used for further analysis to detect sophisticated scanning activity. We define the *darkports* on a given (populated) host as those ports that have not been observed offering any services, and thus are not expected to accept external connection requests (see Section 4.1). We have developed post-scan parsing scripts that process and use heuristics to: (1) classify DCAs into their respective scanning campaigns, and (2) identify and correlate the DCAs that comprise a form of distributed scanning. As one example, we analyze in detail a distributed scan involving the coordination between 14 scanners detected by our technique in one of the network datasets. The raw output from exposure maps supports the rapid development of additional heuristics to identify other types of simple or sophisticated scanning activity (e.g. slow scanning).

3. **Other Applications of Exposure Maps**: the exposure maps technique offers the additional benefit of enabling the following capabilities.

   (a) **Automated response**: the identification of darkports during the construction of the exposure maps provides network-centric knowledge enabling fine-grained automated responses, e.g. to identify and deny specific systems network access when they are found to be both performing scanning activity and thereafter trying to access a legitimate service in the network (common behavior for auto-rooters and scanning worms [44]). This introduces the ability of selective automated response: a focused real-time active response option that limits the introduction of new access control rules to exclusively deny only those scanning systems known to be directly threatening network assets (i.e. those known to be targeting actual services offered by the network). We emphasize the subtle point, that (using our technique) systems that scan for services not offered by the network may be simply identified (i.e. scan recorded) but otherwise ignored (e.g. no access control rule need be introduced to block the associated source IP address). This ability to initiate selective automated response reduces network configuration changes, complexity errors (e.g. by avoiding a dramatic increase in router/firewall rules, and possibly leading to a self-imposed denial of service), and avoids unnecessary performance degradation of network security devices [8, 78].

(b) **Network asset identification and discovery**: the exposure maps technique can be used on both enterprise and backbone networks to logically classify systems into *exposure profiles* that identify and group systems according to the services they offer. We provide a *risk-based* example of how hosts can be grouped into profiles by the services they offer and the perceived threat to the network (e.g. a host that offers a service on a known trojan port would be grouped into the high risk profile). The exposure profiles are configurable and therefore suitable for any network environment. Additionally, we discuss the practical application of exposure profiles and how they can be used to identify malicious network activity (e.g. botnets and worm outbreaks). The technique requires very little computational overhead and easily scales to large enterprise environments or even backbone networks.

(c) **Reconnaissance Activity Assessment (RAA)**: we extend the basic exposure maps concept to include information obtained by recording application banners associated with the IP address and port/protocol of the host offering the service (i.e. an *enhanced exposure map*). Application banners are routinely sent by servers that offer frequently used protocols (e.g. SMTP, HTTP, SSH) responding to client requests. These banners typically contain information such as type of application software and version number. An enhanced exposure map is analogous to an overall catalogue of the host application information (i.e. type and version) divulged by the local network to friend and foe alike. It provides a mechanism to enumerate, record, and report this information leakage in near-real time that enables: (1) confirmation that the applications installed on specific hosts are in compliance with the network security policy and (2) a method to determine what network services (i.e. IP address, port, protocol) and application (e.g. banner, server strings) information was divulged to an adversary as a direct result of specific network scanning campaigns. The framework developed to passively detect the application banners for inclusion in an exposure map is extensible (i.e. using a Bro policy script that contains signatures using special text strings known as regular expressions (`regex`) [52] that can precisely describe search patterns) and can accommodate new or custom application identification. Thus, we provide an answer for the question, “What information has been revealed about the network as a result of a specific detected scanning activity?”

The above contributions are supported by prototype implementations. The three scanning detection techniques mentioned above (described in items 1 (a), 1 (b) and 2 (a)) have been fully implemented (including both UDP and TCP scanning detection) and tested in either a software prototype (for the DNS-based and ARP-based techniques) or as Bro [45] policy scripts (for the exposure maps technique) using commodity platforms. Additionally, we built and tested a new variant of the TRW scanning detection algorithm that augments TRW with our exposure maps technique in a Bro policy.

1.4 Overview of Chapters

The overall structure of the dissertation is as follows. Chapter 2 provides basic background on network discovery techniques, and network scanning strategies. Chapter 3 discusses related work in the field and the current state of the art in network scanning detection techniques. Chapter 4 describes our evaluation approach and methodology for the exposure maps technique including our implementation in a Bro policy. Chapter 5 discusses our evaluation datasets and testing results for all of the exposure maps capabilities including a limited side-by-side comparison with TRW and a modified TRW technique that has been augmented with an exposure maps NEM. Chapter 6 describes our evaluation of our DNS-based scanning detection technique that includes a detailed discussion of our implementation, evaluation methodology, testing results, and limitations. Chapter 7 describes our evaluation of our ARP-based scanning detection technique that includes a detailed discussion of our implementation, evaluation methodology, testing results, and limitations. Chapter 8 summarizes the contributions this dissertation and describes future research directions. Finally, Appendix A contains reference material that includes an acronym list as well as a list of network services with the associated TCP/IP port numbers.

Chapter 2

Background

In this chapter we provide a discussion of passive and active network discovery techniques as well a variety of network scanning strategies.

Some form of reconnaissance activity often precedes an attack. An effective mechanism used by an adversary to remotely probe a network is *port scanning* [16]. A *port scan* can be defined as sending packets to a particular IP/port combination to elicit a response in an attempt to discover active systems in the network and the particular services they offer. Even in the instance where no response is given from the host being port scanned, the adversary still learns something about the host or the network where it resides. Namely, the host is either unreachable due to network constraints (e.g. not accessible remotely, behind a firewall) or the service being scanned is simply not offered. Port scans are directed against a host using either the UDP or TCP protocol on one or more of the 2^{17} possible ports (i.e. 2^{16} TCP ports and 2^{16} UDP ports). Herein when the terms scanning or scan are used in this thesis we are referring to port scanning. Additionally, we refer to a series of connection attempts (i.e. one or more) to IP/port combinations in which there is insufficient evidence to determine if the activity is a scan or simply the result of legitimate connection attempts as *ambiguous connection attempts.*

We define how exposure maps detects scanning activity in Section 4.1.1 and describe two heuristics we developed for analyzing Darkport Connection Attempts (see Section 4.1.1, page 43) generated by the exposure maps technique for use in our side-by-side comparison with TRW and the detection of distributed scanning activity in Sections 5.1 and 5.1.2 respectively. We define how the modified TRW and TRW techniques classify network activity as scanning activity in Sections in 4.1.5 and 3.1 respectively. Finally, we define how the DNS-based scanning detection (i.e. anomalous DNS activity) and ARP-based scanning detection (i.e. anomalous ARP activity) techniques classify network activity as scanning activity in Sections 6.1 and 7.1 respectively.

2.1 Passive and Active Network Discovery Techniques

Network scanning is a form of *active* scanning. The term active is used to describe this activity as some stimulus (i.e. packet) is injected in the network to generate a response from the recipient. Active scanning is a technique employed by malicious adversaries to probe a network. However, active scanning has legitimate uses as a part of a robust network security policy that incorporates vulnerability scanning. Active scanning allows a network operator to discover the open services in the network so they can be checked for known vulnerabilities. For instance, Nessus [41] is an active vulnerability scanner that can discover the type of operating system, applications, databases, and network services running on individual hosts. This information can be checked against a database of known vulnerability information to determine if a discovered host is potentially vulnerable to any of these known attacks. In contrast, *passive* service discovery is the process of monitoring network layer traffic to detect network topology, offered services, and applications. Passive service discovery does not generate any network traffic between the servers and clients in a network.

Remote operating system fingerprinting is the action of trying to remotely determine the operating system of a particular host of interest. For instance, Nmap [21] is an active OS fingerprinting tool that sends packets to a host so that any responses (or lack of responses) can be analyzed. The responses to these sequences of packets form a signature or *fingerprint* for the remote operating system that can be compared against a signature database of known operating system versions. Operating system fingerprinting takes advantage of the observation that each operating system's network stack (i.e. software that implements the TCP/IP protocol) has slight variations in the way it responds to certain packets. These variations offer the ability to remotely determine the type of operating system. For instance, one OS fingerprinting test is to send a TCP SYN packet with no flags set (i.e. known as a NULL packet) to the host being scanned. The type of response (or lack of a response) to this packet gives insight about the type of operating system being scanned. A series of additional similar *tests* are performed concurrently (using other forms of unusual packets) and once the results are correlated and analyzed, the type and version of the operating system is usually revealed. In contrast, p0f [81] is a passive operating system fingerprinting tool. It can determine characteristics such as the presence of a firewall or proxy, distance (hop-count) to the physical system, host up-time, and the link speed of the remote host. p0f does not generate any packets on the network it is monitoring. Instead, p0f relies on extracting features exclusively from the passive observation of network activity between hosts of interest.

Application fingerprinting is the action of trying to remotely determine the applications or services running on a particular host of interest. Servers routinely send information about the applications they are running to client systems during normal connection activities. The initial text sent by servers during a connection attempt is known as a *banner*. The act of harvesting banners (i.e. *banner grabbing*) during passive or active identification of network systems and their applications is a well understood concept. For instance, banner grabbing would be routinely performed during vulnerability testing (e.g. penetration testing) of the network. The software versions advertised in application banners can identify potential security issues if it is determined that the software version contains known vulnerabilities.

An examination of the Server field in the HTTP response header, such as `Apache 2.0.54 (Debian GNU/Linux)`, provides an example of how banner grabbing can be used to identify HTTP servers. This response exhibits the type of server software, software version, and underlying operating system. The purpose of an application banner is to facilitate determining how a client system can best interact with the server based on the applications it is running. This information also divulges to an attacker potential vulnerabilities in the system associated with the specific version of the software.

Passive Asset Detection System (PADS) [61] is signature-based software used to passively detect network assets using application fingerprinting. It attempts to provide an accurate and current listing of the hosts and services offered on the network. It utilizes the TCP, ARP, and ICMP protocols to perform its signature matching.

One of the methods it uses to perform application layer fingerprinting is banner grabbing. Trickler is another publicly available (upon request) software tool to do passive network characterization [34]. It can passively fingerprint hosts without the use of signatures and track the state of tens of millions of hosts on commodity hardware.

Considerations for Passive and Active Techniques. Active scanning techniques are well suited to find the open services currently on a network. However, these techniques are not without their drawbacks. The results from an active scan can only be considered a *snapshot* in time of the services being offered on the network and may quickly become out of date and inaccurate. Furthermore, the scans generated by these techniques can be invasive and in some instances cause service interruptions due to the type of packets (e.g. malformed, unlikely flag combinations, etc.) that may be sent when trying to identify a host or service. In contrast, passive network discovery techniques continuously monitor the network providing a near real-time view of the offered services and active hosts. As no packets are injected in the network, there is no risk of unintentional service disruption. Passive monitoring also has the advantage of being able to find intermittently offered or protected services that are often missed by active probing (e.g. behind a firewall) [6]. However, passive monitoring techniques are much slower at detecting inactive hosts and transient services. Additionally, passive network discovery techniques can identify hosts and variations in versions of the services they offer only if it is discernible in network traffic (i.e. the particular software version transmits some type of unique pattern). For instance, if a patch was applied to a new version of an application that did not change its behavior on the network, it would not be possible to determine if the host was using the old or new version of the software. It is important to note that active scanning techniques will also have this limitation if a type of probe cannot be developed to differentiate between software versions. Finally, there have been techniques proposed to defeat OS fingerprinting [62] that could be extended to application fingerprinting. In this scheme, a network *scrubber* is used in-line to modify packets sent by local network hosts to remote systems thus obfuscating the features required by OS fingerprinting tools.

2.2 Network Scanning Strategies

Network scanning activity in an enterprise network can be characterized as occurring: local to local (L2L), local to remote (L2R), and remote to local (R2L).

L2L network scanning refers to a host that scans systems within the boundaries of the enterprise network in which it resides (see Figure 2.1(c)). Topological scanning worms frequently employ this type of scanning strategy. In larger enterprise networks, it is not unusual for network segments to be either logically or physically separated. In fact, an enterprise network may be comprised of several distinct subnets (often referred to as *cells*). This separation can occur for a variety of reasons including security, ease of administration, and geographic location. Figure 2.1(a) shows a network containing three logical cells. L2L network scanning activity can occur within or between network cells.

L2R network scanning refers to a host within administrative control of the enterprise network scanning systems outside the network boundary (see Figure 2.1(d)). In this context, the scanning host is performing network reconnaissance against external systems.

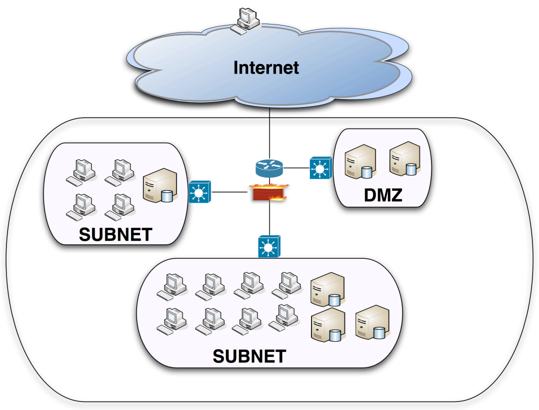
R2L network scanning refers to scanning activity directed at a network by a host outside the target network's administrative control (see Figure 2.1(b)).

The majority of publicly available research to date has been focused on detecting R2L network scanning activity. We have developed scanning detection strategies that detect R2L as well as L2L, and L2R scanning activity.

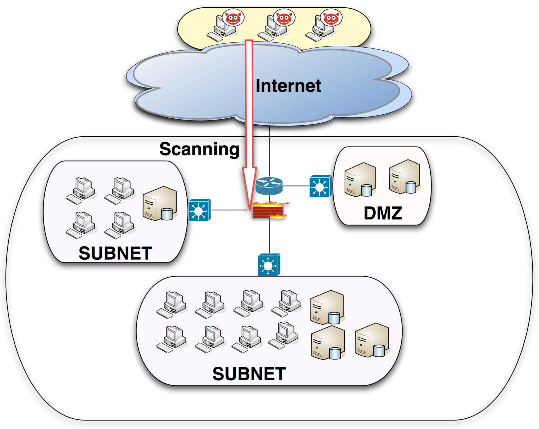
The following two subsections describe network scanning approaches that can be broadly characterized into two categories: wide-range reconnaissance, and target-specific reconnaissance.

2.2.1 Wide-range Reconnaissance

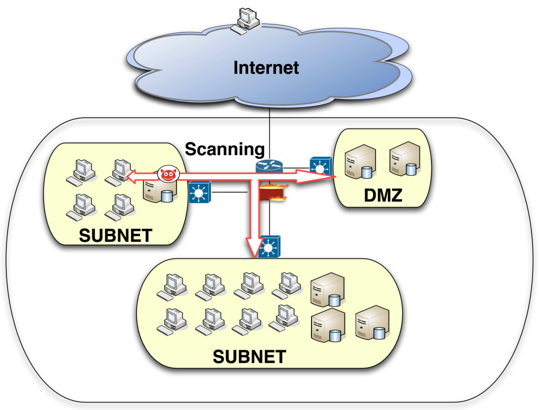
Wide-range reconnaissance can be defined as the rapid scanning of large blocks of Internet addresses in the search for a specific service or vulnerability. Typically, there is little human interaction in this type of reconnaissance. This is characteristic of *auto-rooters* [66] and worm propagation. Auto-rooters are composite tools that augment basic port scanning functionality by launching an attack as soon as an open



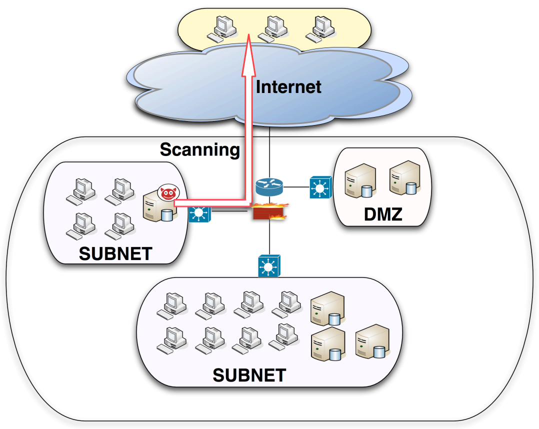
(a) Network Cells.



(b) Remote to Local (R2L) Scanning Activity.



(c) Local to Local (L2L) Scanning Activity.



(d) Local to Remote (L2R) Scanning Activity.

Figure 2.1: Network Scanning Activity.

port is located on a target system [4]; they are often used for the rapid enrollment of vulnerable systems into botnets of tens or hundreds of thousands of compromised systems [5]. Simple scanning worms propagate by indiscriminately probing the Internet as rapidly as possible to locate and infect vulnerable systems. Scans from auto-rooters and scanning worms can usually be attributed to the *true* source as the scans themselves are the first stage of the actual exploit attempt (e.g. a response, from the target, to a TCP SYN connection request will start the exploit in the same session). The next section explores worm propagation strategies in detail.

2.2.2 Target-specific Reconnaissance

In contrast to such indiscriminate scanning, skilled adversaries may go to considerable lengths to mask their activities. Numerous sophisticated scanning techniques allow stealthy, focused scanning of a predetermined target (host and/or network); some of these make attribution to the scanning source impractical, rendering most current scanning detection techniques ineffective. The following techniques belong to this category.

Indirect scanning occurs when an attacker uses one system (or systems) to scan a target and another system to attack the target. This separation defeats attribution attempts. If the scanning activity from the scanning system is detected (e.g. blocked at a network router, or by system administrator intervention), the attacker simply uses another scanning system. A slightly more sophisticated variation uses *throw-away* scanning systems, i.e. previously compromised systems that have little value to an attacker other than being able to provide a disposable platform to perform tasks. Any scanning activity traced back to the source, will be attributed to the owner of the compromised system.

Distributed scanning occurs when multiple systems act in unison using a divide and conquer strategy to scan a network or host of interest. Typically, one system will act as a central node and collect the scanning results from all participating systems. Distributing the scanning activity reduces the scanning footprint from any single system and thus reduces the likelihood of detection. An extreme version of distributed scanning involves an attacker using a botnet (see below) to scan a target in a coordinated manner resulting in very stealthy scans.

Botnet scanning occurs when a collection of compromised systems (bots or zombies) are used to scan a target. A botnet can provide an attacker with, in essence, an unattributable method of reconnaissance. For instance, consider a botnet owner that has an exploit capability against a network service. A botnet of approximately 65,000 systems would be able to scan an entire Class B network for this service by sending a single packet from each bot (each with a unique IP address). In this example, even if it were possible to correlate this activity to a single scanning campaign, it still would not reveal the true adversary as the bots are simply zombie participants.

Idle scanning [57] allows an attacker to port-scan a target without sending a single packet from the attacker's own system.[1] The attacker first sends a SYN packet to the port of interest on the target host spoofing the source address of the packet with the IP address of an innocent system (hereafter referred to as a bot). If the port is open, the target responds to the bot with a SYN ACK. The bot does not expect this unsolicited SYN ACK packet so it responds with a RST packet to the target and increments the 16-bit identification field (IPID) it includes in its IP header. The attacker then sends a SYN packet to the bot and observes the IPID field of the RST packet the bot sends back. If the IPID has been incremented, the port on the target was open. Idle scanning utilizes side-channel communication by redirecting the scan and bouncing it off a third-party system. Most scanning detection algorithms will erroneously identify the third-party system as the scanner.

Low and slow scanning occurs when an attacker slowly scans a target host or network (e.g. a single scanning campaign may take days, weeks or months). Slow scans may blend into the network *noise* never exceeding detection thresholds or exhausting detection system state.

2.2.3 Worm Propagation Methods

A *computer worm* is a program that can self-propagate across a network by exploiting vulnerable systems [69]. Worms are typically classified based on two attributes: methods used to spread, and the techniques used to exploit vulnerabilities. Every

[1]See also: Idle Scan and related (IPID) games, http://www.insecure.org/nmap/idlescan.html

vulnerability a worm can exploit allows it to self-propagate across a network. Most worms propagate by using indiscriminate scanning of the Internet to identify vulnerable systems. As revealed by Slammer [39], the faster a worm can locate systems the more rapid the infection rate. Staniford et al.' s study [65] used empirical data from actual worm outbreaks to derive a common effective propagation strategy, the random constant spread (RCS) model, wherein a worm randomly scans through the entire Internet IPv4 address space, of 2^{32} systems, searching for vulnerable systems.

Traditional Propagation Methods. The limiting factors which dictate how fast a worm can spread are [68]: (1) the rate of scanning used to detect vulnerable systems, (2) the population of vulnerable systems, (3) the time required to infect vulnerable systems, and (4) its resistance to countermeasures. The spread of a random scanning worm can be described in three phases [82]: the *slow spreading phase*, *fast spreading phase*, and *slow finishing phase*.

In the slow spreading phase, the worm is building up an initial base of infected systems. Although it is infecting systems at an exponential rate, the small initial population limits the propagation speed. Once a certain threshold of infected hosts is reached, the worm begins the fast spreading phase. Models derived from actual worm data indicate that this threshold is approximately 10,000 systems [65]. Worms can use a number of different techniques to propagate including [65]:

1. *Random Scanning.* The Internet IPv4 address space consists of 2^{32} unique IP addresses. Using a pseudo random number generator, the worm randomly scans the Internet address space searching for systems to infect.

2. *Subnet Scanning.* A subnet is a range of logical IP addresses within a defined network address space. Using the host's own IP address as a reference, the worm scans the IP addresses in that subnet. Generally, systems within a subnet will be located physically close to one another and have a similar security stance.

3. *Topological Scanning.* Instead of performing random or subnet scanning, the worm uses information resident on the host to find vulnerable systems. This information includes URL caches, peer-to-peer connections, trusted network connections, and email addresses from address books on the host machine.

4. *Mass email.* Using host-resident emailers, worms can utilize information as in topological scanning, for example harvested email addresses from existing email address books, the inbox of the email client, and web page caches. Copies of the worm are then sent to all the hosts corresponding to the harvested email entries.

5. *Network Share Traversal.* Through shared network drives, systems often have access to directories on other systems. By placing itself on a host that is part of a shared system, a worm can use this shared access to infect other systems. The worm can also take a more active role and change permissions on directories or add guest accounts. This technique is also related to topological scanning.

6. *Web-based Attacks.* There are a multitude of web server attacks available that can be used by worm writers to infect systems [50]. Once the worm has infected the system, it can append malicious code to specific or random web pages. Systems that access these modified web pages with a vulnerable web browser will be infected.

7. *Hit-list Scanning.* A hit-list is a list of vulnerable systems that are high-probabillity-of-success candidates for infection. Typically, a hit-list is generated by previous reconnaissance activities such as: network scanning, web surveys, DNS queries to obtain IP addresses, and web spiders. A hit-list is used to allow a worm to rapidly spread in the first few minutes. This increases its virulence and its chances of survival.

8. *Permutation Scanning.* Random scanning of the Internet for vulnerable hosts can be inefficient because many addresses may be probed multiple times. One way to address this limitation is to have the worms share a common pseudo random permutation of the Internet address space. Infected systems can start scanning at a specific point in the permutation selected in such a way as to avoid unnecessary duplication in scanning between infected systems. If the worm detects an infected system within its scanning permutation order, it simply picks a new random point in the permutation and begins scanning again. Thus, no communication between infected systems is required to reduce the amount of

redundant reinfection and this technique also imposes a measure of coordination on the worm.

Items 7 and 8 are hyper virulent worm propagation strategies described by Weaver et al. [65]. Prior to 2004, these strategies were apparently not implemented by worm writers. The analysis of the Witty worm [60] suggests that due to the propagation characteristics, simultaneous scanning likely occurred due to the use of a hit-list or because of a timed release from a group of previously compromised systems.

Chapter 3

Related Work

In this chapter, we review the relevant literature on techniques and methods to perform network scanning detection for an enterprise network. Section 3.1 contains an analysis of the Threshold Random Walk (TRW) scanning detection algorithm that includes a description of a specific implementation of the algorithm by Jung et al. [29]. Sections 3.2 and 3.3 respectively, discuss a variety of additional external network scanning detection (i.e. R2L) strategies and internal network scanning detection (i.e. L2L and L2R) strategies.

3.1 External Network Scanning Detection Strategies – TRW

Jung et al. [29] developed an algorithm called Threshold Random Walk (TRW), to identify malicious scanning activity from remote hosts. We summarize their work here. The approach uses *sequential hypothesis testing* to evaluate each new connection request. They based this algorithm on the observation that scanners are more likely to try and access hosts and services that do not exist than legitimate remote hosts. Thus, if a host tries to connect to a system and the destination exists, then there is support for the hypothesis that the source is benign. If the destination does not exist, this is used as support for the hypothesis that it is more likely a scanning system. Once there is enough evidence gathered through observation of ongoing network activity, one of two hypotheses is either accepted or rejected.

Specifically, the algorithm evaluates two possibilities, H_0 and H_1, where H_0 is the hypothesis that the remote source of the connection attempt (r) is benign and H_1 is the hypothesis that r is a scanner. Given a remote host r, let Y_i be the random variable that represents the outcome of the connection attempt by r to the i^{th} distinct local host, where

$$Y_i = \begin{cases} 0 & ; \text{ if the connection attempt is a success} \\ 1 & ; \text{ if the connection attempt is a failure} \end{cases} \tag{3.1}$$

If we assume that conditional on the hypothesis H_j, the random variables $Y_i, i = 1, 2, ...$ are both independent and identically distributed, then variable Y_i can be expressed as

$$\begin{aligned} Pr[Y_i = 0|H_0] = \theta_0; \quad Pr[Y_i = 1|H_0] = 1 - \theta_0 \\ Pr[Y_i = 0|H_1] = \theta_1; \quad Pr[Y_i = 1|H_1] = 1 - \theta_1 \end{aligned} \tag{3.2}$$

We would expect θ_0 to be relatively high, and θ_1 to be relatively low (as discussed further below). As each new connection attempt is observed, the *likelihood ratio* for the remote host is defined as [29]

$$\Lambda(r) = \prod_{i=1}^{n} (\frac{Pr[Y_i|H_1]}{P_r[Y_i|H_0]}) \tag{3.3}$$

where Y_i is the collection of events observed thus far involving the specific remote host r, and n is the total number of observed connection attempts by r to the local network. The likelihood ratio is compared to a lower threshold, η_0, and an upper threshold, η_1. If $\Lambda(r) \leq \eta_0$ then hypothesis H_0 is accepted. If $\Lambda(r) \geq \eta_1$ then hypothesis H_1 is accepted. After a certain number of connection attempts or a timeout period is observed if neither η_0 or η_1 is reached, the remote host is classified as undetermined, i.e. it is not possible to determine if it is a scanner or benign (see Figure 3.1).
The two hypotheses (H_0, H_1) can generate four possible outcomes.

1. True Positive: this decision occurs when H_1 is selected and H_1 is true.
2. False positive: this decision occurs when H_1 is selected and H_0 is true.
3. False negative: this decision occurs when H_0 is selected and H_1 is true.
4. True Negative: this decision occurs when H_0 is selected and H_0 is true.

The TRW probability of detection variable, P_D, and the false positive probability, P_F, are used to bound the performance conditions of the algorithm. The detection probability, P_D, is the probability that H_1 is selected when H_1 is true while the false

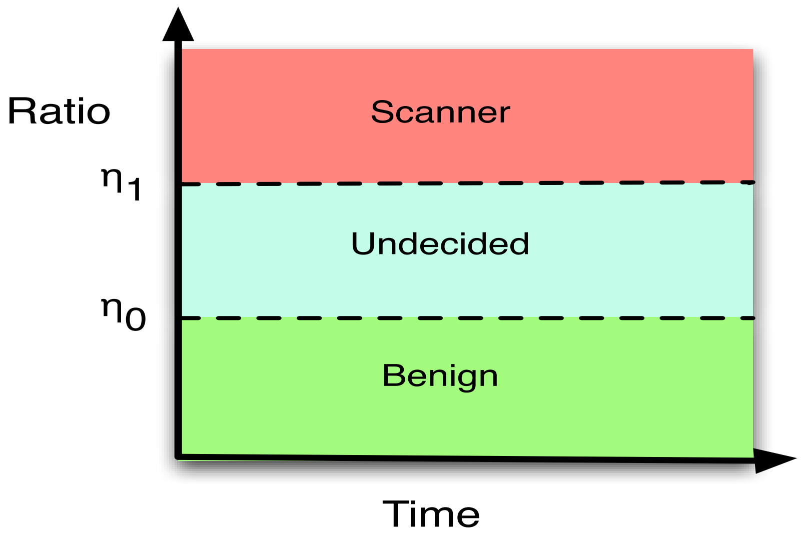


Figure 3.1: Using Likelihood Ratio to Classify Remote Hosts.

positive probability, P_F is the probability that H_1 is selected and H_0 is true. The upper threshold (η_1) that supports hypothesis H_1 is

$$\eta_1 = \frac{P_D}{P_F} \tag{3.4}$$

When P_D and P_F are replaced by the user defined values β and α respectively, η_1 is defined as

$$\eta_1 := \frac{\beta}{\alpha} \tag{3.5}$$

The lower threshold (η_0) that supports hypothesis H_0 is

$$\frac{1 - P_D}{1 - P_F} = \eta_0 \tag{3.6}$$

When P_D and P_F are replaced by the user defined values β and α respectively, η_0 is defined as

$$\eta_0 := \frac{1 - \beta}{1 - \alpha} \tag{3.7}$$

The average number of connections a remote host needs to make (N) before a determination of whether it is benign or a scanner can be made is a function of α, β,

θ_0, and θ_1. Specifically, the expected number of observations required to determine if a remote host is benign, $E[N|H_0]$, or a scanner, $E[N|H_1]$, is defined as [29]

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

$$E[N|H_1] = \frac{\beta \ln \dfrac{\beta}{\alpha} + (1-\beta) \ln \dfrac{1-\beta}{1-\alpha}}{\theta_1 \ln \dfrac{\theta_1}{\theta_0} + (1-\theta_1) \ln \dfrac{1-\theta_1}{1-\theta_0}} \tag{3.8}$$

The alert threshold (η_0 and η_1) is defined based on both the desired detection and false positive/negative rates. The typical assignment for a desired false positive rate (α) is .01 and .99 for the desired detection rate (β) [29]. The θ_0 and θ_1 variables define the calculations of the likelihood ratio based on whether a successful connection from a remote host is from a benign or scanning host respectively.

The value chosen for θ_1 should be dependent on the network being monitored. A reasonable setting for θ_1 would be based on a ratio of the number of remotely accessible active hosts in the network (i.e. density of remotely accessible hosts) to total local network address space. For instance, in a quarter class C network (not including broadcast addresses) with 3 remotely accessible servers, a value of $\theta_1 = .04838$ (i.e. 3/62) may be appropriate. It is more subjective as to what the value θ_0 should be set. However, given that θ_0 is the probability that a benign host makes a successful connection attempt it should be set relatively high. In the original paper [29], the authors set the value of $\theta_0 = .80$ and $\theta_1 = .20$ based on their analysis: that very few benign hosts made connections to more than four distinct IP addresses; the observed differences in the network behavior of scanning and benign hosts within the network datasets; and the knowledge of the host density of the network.

Using these same values if a remote host made four failed connection attempts, in the absence of any successful connection attempts, it would be considered a scanner [29]; there was an additional constraint for this specific implementation in that these failed connection attempts had to occur against unique hosts. Note that from equation (3.3), the smallest positive integer x such that $\Lambda(r) = (\frac{1-\theta_1}{1-\theta_0})^x \geq \eta_1$ is $x = 4$. It was argued that requiring the observation of at least four failed connection attempts to unique hosts in the network before a remote host could be classified as a scanner would reduce the number of false positives that a lower threshold would produce due to either peer-to-peer (P2P) or backscatter traffic [40]. In the case of a remote host that makes both successful and failed connection attempts to the local network, the likelihood ratio will either be reduced or increased by an amount dependent, in part, on the values of θ_0 and θ_1. The calculations of the likelihood ratio, and the comparisons to the thresholds, will continue until some threshold (η_0 or η_1) is reached thereby selecting a hypothesis, or the connection pattern history of the remote host does not allow the algorithm to reach a decision on whether the remote host is benign or a scanner (e.g. an insufficient number of connection attempts are observed or the connection patterns are such that no decision on a hypothesis can be made). It should be noted that the summing of values $\theta_0 = .80$ and $\theta_1 = .20$ to 1 is purely coincidental and leads to symmetries (e.g. in terms of the number of failed or successful connection attempts to declare a remote host is a scanner or benign) that do not hold in general.

TRW, as discussed above, has been implemented as a scanning detection algorithm in Bro. Bro [9] is an open source intrusion detection system developed by Paxson. Bro passively monitors a network link and uses an event engine to generate a variety of events based on the network traffic it intercepts. Events are then processed by user-supplied policy scripts to perform both network and security analysis in near real-time. Policy scripts contain event handlers that can perform various actions based on the corresponding event generated. The event handlers can generate and maintain global state information as well as specify the actions Bro should take (e.g. record activity to log file, update data structures, generate an alert) in response to network events. Bro inspects and groups packets into connections for analysis. This model is a natural fit for the TCP protocol. The UDP and ICMP protocols are connectionless but nonetheless are also modeled as connections using a *flow-like* definition. Specifically, a flow can be described as a set of TCP or UDP packets that share the same connection parameters (i.e. source IP, destination IP, source port, and destination port) indicating they are part of the same connection.

The `trw-impl.bro` policy implements the TRW algorithm by Jung et al. [29]

for the TCP protocol. In order to understand how TRW is implemented as a Bro policy, we now discuss in detail relevant aspects of Bro as well as the high-level TRW implementation details found in the `trw-impl.bro` policy.

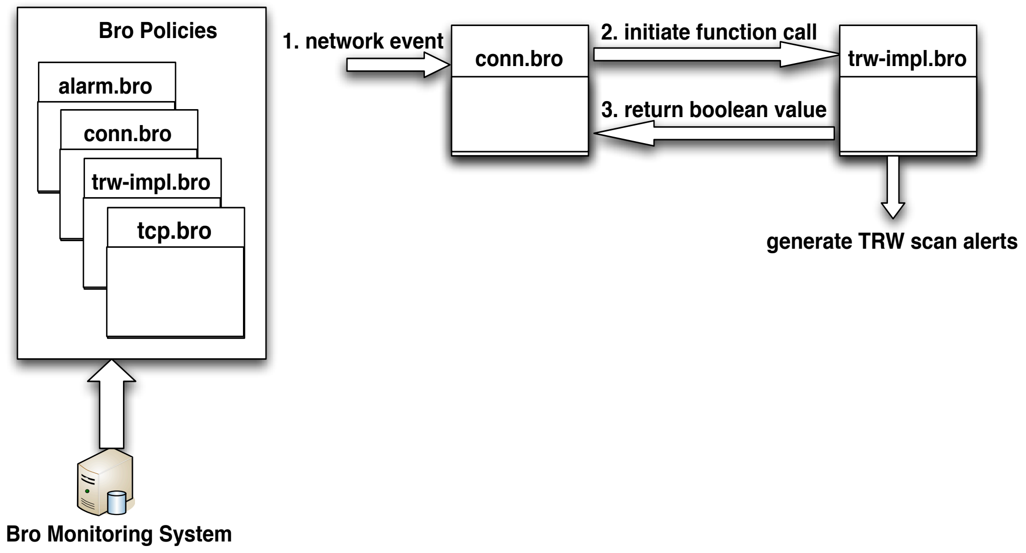


Figure 3.2: trw-impl.bro Policy Interaction.

The TRW algorithm is implemented as a function (i.e. `trw-impl.bro`) called when any of the following events are handled in the `conn.bro` policy: `event connection _established`, `event connection_attempt`, or `event connection_rejected` (see Figure 3.2). When these events occur, connection information (i.e. a Bro-defined connection record) as well as a boolean variable `reverse` is passed to the TRW function for evaluation. The `reverse` variable, if true, indicates that a TCP connection has been established due to the detection of a SYN ACK packet but the associated SYN packet that should have started the three-way TCP handshake has not been observed. Although this is a natural possibility, it may also be as a result of SYN ACK scan (i.e. a scan that consists of packets with the SYN ACK flag set) or more likely backscatter traffic [40]. Ordinarily, Bro interprets the end point system that sends the SYN ACK packet as the responding host. In this case, the end point system that sent the SYN ACK packet is actually the originator of the connection (in the context of responding and originating systems). Bro switches the systems roles accordingly. The `conn.bro` policy is primarily responsible for generating one line summaries of

Table 3.1: Bro Connection States (adapted from [10]).
*indicates the state is associated with a failed connection attempt.

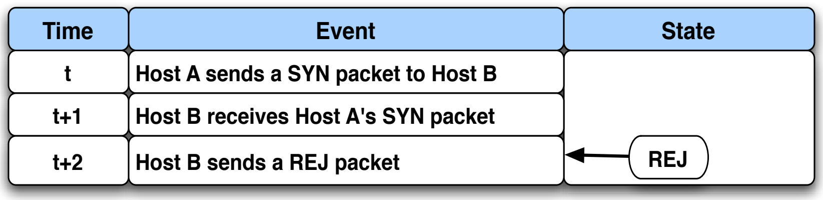
State Name	Description
S0*	SYN packet detected, indicating a new connection but no reply was observed.
S1	Connection established, not terminated.
SF	Normal connection establishment and termination.
REJ*	Connection attempt rejected.
S2	Connection established and close attempt by originator seen (but no reply from responder).
S3	Connection established and close attempt by responder seen (but no reply from originator).
RSTO*	Connection established, originator aborted (sent an RST).
RSTR	Connection established, responder aborted.
RSTOS0	Originator sent a SYN followed by a RST, no associated SYN ACK packet was observed from the responder.
RSTRH	Responder sent a SYN ACK followed by an RST, we never saw a SYN from the (purported) originator.
SH	Originator sent a SYN followed by a FIN, we never saw a SYN ACK from the responder (hence the connection is *half* open).
SHR	Responder sent a SYN ACK followed by a FIN, no SYN from the originator.
OTH*	No SYN seen, just midstream traffic (a *partial connection* that was not later closed).

observed connection information. Each connection is assigned a state as defined in Table 3.1.

Figure 3.4 shows the sequence of events for the establishment and termination of a *normal* TCP connection. Mapped within the state portion of this figure are seven of the associated states identified in Table 3.1 at the point in the connection attempt where they would be generated if the connection ended at the associated time (t). For instance, the state S0 is generated when a SYN packet is observed at time (t)but no other packets for this connection are seen. State S1 is generated as a result of a TCP connection being fully established (i.e. three-way handshake was completed from time t until time t+5) but no subsequent termination of the connection was observed i.e., FIN and ACK packets expected between time ($t+7+n$) and ($t+13+n$) did not occur. The value $n \geq 0$ indicates some arbitrary amount of time has elapsed during data transfer which depends on the specific connection. A state of SF indicates that a complete TCP session was successfully established and terminated, i.e., all events from (t) until (t+13+n) occurred.

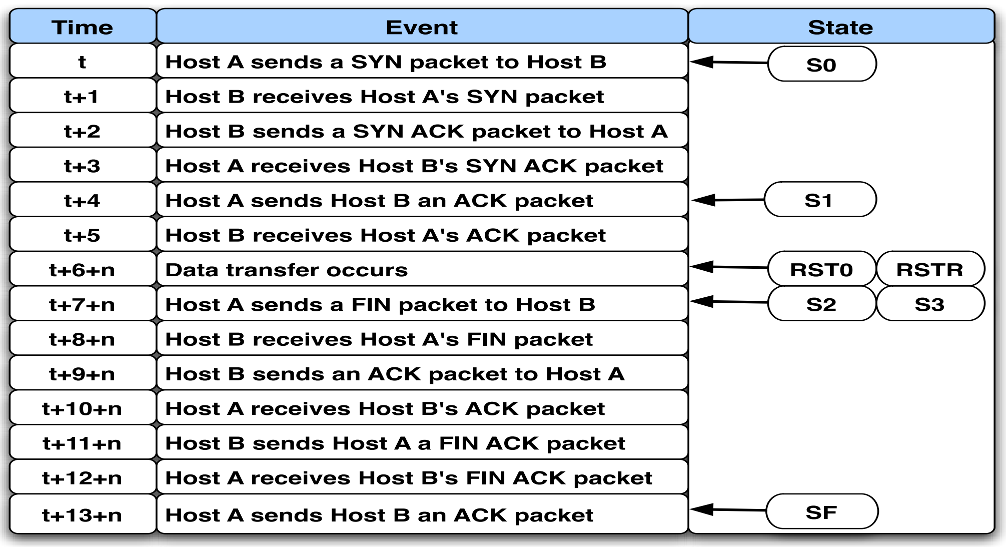
Figure 3.3, shows the sequence of events that occur during a rejected TCP connection. In this case, Host B rejects Host A's TCP connection request and sends it a packet with the REJ flag set.

The remaining states RSTRH, RSTOS0, SH, SHR, OTH from Table 3.1 are not present in Figures 3.4 and 3.3. These states are generated as a result of some part of the *normal* connection activity not being observed in the network traffic. For instance, the OTH state indicates that the three-way handshake of the connection was never observed in the network traffic.



Time	Event	State
t	Host A sends a SYN packet to Host B	
t+1	Host B receives Host A's SYN packet	
t+2	Host B sends a REJ packet	REJ

Figure 3.3: Rejected TCP Connection.



Time	Event	State
t	Host A sends a SYN packet to Host B	S0
t+1	Host B receives Host A's SYN packet	
t+2	Host B sends a SYN ACK packet to Host A	
t+3	Host A receives Host B's SYN ACK packet	
t+4	Host A sends Host B an ACK packet	S1
t+5	Host B receives Host A's ACK packet	
t+6+n	Data transfer occurs	RST0 RSTR
t+7+n	Host A sends a FIN packet to Host B	S2 S3
t+8+n	Host B receives Host A's FIN packet	
t+9+n	Host B sends an ACK packet to Host A	
t+10+n	Host A receives Host B's ACK packet	
t+11+n	Host B sends Host A a FIN ACK packet	
t+12+n	Host A receives Host B's FIN ACK packet	
t+13+n	Host A sends Host B an ACK packet	SF

Figure 3.4: Normal TCP Connection.

The S0, REJ, OTH, and RSTO states all indicate some form of connection failure has occurred (see Table 3.1). The `trw-impl.bro` policy uses these connection states to determine if a connection attempt has succeeded or failed. Recall that the TRW algorithm is based on the observation that a scanner's connection attempt is more likely to fail than a benign host's connection attempt.

The `trw-impl.bro` policy script attempts to classify hosts into five categories (see Figure 3.5) based on the connection behavior they exhibit in accordance with the pseudo code algorithm described in Figures 3.6 and 3.7. Specifically, a host that exceeds a predetermined threshold of failed connection attempts it is considered a *scan source*. The individual hosts within the local network that were scanned by the scan source are known as *failed locals*. A host that exceeds a predetermined threshold of successful connection attempts is considered to be a *benign source*. The individual hosts within the local network that participated in the connections from the benign source are known as *successful locals*. Finally, an external host that has been accessed by an internal host prior to initiating a connection is considered a *friendly remote*.

Figure 3.6 (derived from [9]) shows the inputs, parameters, and associated default values for the `trw-impl.bro` policy script. The following four parameters are used to classify hosts based on their activity as described above as well as to determine the overall sensitivity of the detection rate and the tolerated false positive rate:

1. `target_detection_probability`: the probability that a scan will be detected.

2. `target_false_positive`: the tolerated false positive probability.

3. `theta_zero`: the probability that a given remote host's connection will succeed.

4. `theta_one`: the probability that a scanner's connection attempt will succeed.

We now discuss how each of these parameters are set and provide a complete description of the pseudo code representation of the TRW algorithm found in Figure 3.7 (adapted from [9]). The connection record `c` for a particular connection event provides the source IP address, destination IP address, destination port, and connection state information used in the `trw-impl.bro` policy script. First, at line 1, a check is made to determine if the source IP address is part of the local network. If it is, the

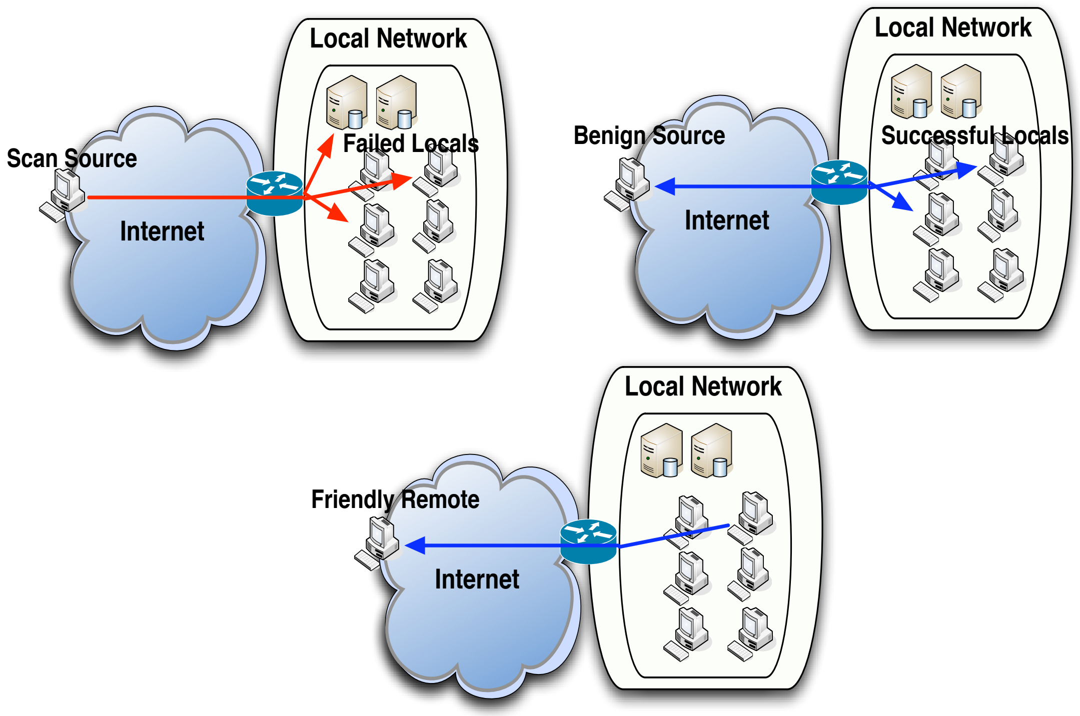


Figure 3.5: Host Classification Possibilities for the trw-impl.bro Policy Script.

destination IP is checked to see if (1) it is not included in the `scan_scources` list, (2) the destination port is not a member of the `triggered_outbound_services` set, and (3) the connection state is not equal to `OTH`. The `scan_sources` set contains the set of hosts that have been identified as scanners. If these checks are *true*, the destination IP address is added to the `friendly_remotes` list and the function returns with the boolean value of false (indicating this was not scanning activity). These checks serve to identify local host initiated connections to remote hosts in which case the remote hosts is considered to be *friendly*. There is one exception to subsequent inclusion in the `friendly_remotes` set that occurs when a local host attempts to connect to a destination port listed in the `triggered_outbound_services` list.

The `triggered_outbound_services` set contains a list of services (e.g. finger, ident, FTP) known to make a host initiate a connection in response to a previously received connection. For instance, the ident service [27] was designed to identify the user or users of a TCP connection. The service normally runs as a server daemon

Figure 3.6: trw-impl.bro Pseudo Code - Inputs and Parameters.

```
Inputs:
1:  c: connection record.
2:  state: the observed end state of connection.
3:  reverse: boolean.
Parameters:
4:  target_detection_prob = 0.99 :probability of detection.
5:  target_false_positive_prob = .01 :probability of generating a false
    positive.
6:  failed_locals = set of IP addresses indexed by the IP address pair of the
    scanner IP address and a unique host IP address it has scanned.
7:  successful_locals = set of IP addresses indexed by the IP address pair
    of a remote host IP address and the unique host IP address it has made
    successful connections to.
8:  lambda = table of TRW values associated with each host for comparison
    with the TRW threshold. Indexed by the IP addresses of remote hosts.
9:  num_scanned_locals = the size of the set of unique local host IP addresses
    a scanning system has scanned.
10: theta_zero = 0.8 :probability that a legitimate remote's TCP connection
    attempt to a unique host will succeed. Depends on the TRW model and the
    local network topology.
11: theta_one = 0.2 :probability that a scanner's TCP connection attempt to a
    unique host will succeed. Depends on the TRW model and the local
     network topology.
12: eta_one = .99 :based on equation (3.5) using the values of
    target_detection_prob and target_false_prob.
13: eta_zero = 0.01 :based on equation (3.7) using the values of
    target_detection_prob and target_false_prob.
14: triggered_outbound_services = ident, finger, 20/TCP :list of
    services exempt from scanning detection.
15: friendly_remotes = set of IP addresses of the remote hosts that have been
    accessed by internal hosts.
16: scan_sources = set of IP addresses of the remote hosts that have been
    identified as scanners.
17: benign_sources = set of IP addresses of the remote hosts that have made
    previous successful connection attempts.
18: established = false :initializing established variable.
19: flag = 0 :initializing flag variable.
```

Figure 3.7: trw-impl.bro Pseudo Code - Main Body of Function.

Main:

```
if {(src_ip) is a local network address}
    if {(dest_ip) not in scan_sources}
        and {dest_port not in triggered_outbound_services}
        and {state <> OTH}
            add (dest_ip) to friendly_remotes
            return false
    endif
endif
if {(src_ip) in scan_sources}
    return true
endif
if {state is one of S0, REJ, OTH, RSTO}
    established := false
endif
if {not established}
    if {(src_ip, dest_ip) not in failed_locals}
        flag := 1
        add (scr_ip, dest_ip) to failed_locals
        ratio := (1- theta_one)/(1- theta_zero)
    endif
elseif {(scr_ip, dest_ip) not in successful_locals}
    flag := -1
    add (src_ip, dest_ip) to successful_locals
    ratio := theta_one/theta_zero
endif
if {flag = 0}
    return false
endif
updated_lambda[src_ip] := lambda[src_ip] * ratio
if {updated_lambda[src_ip] > eta_one}
    add (src_ip) to scan_sources
    raise alert scan detected
    return true
endif
if {updated_lambda[src_ip] < eta_zero}
    add (src_ip) to benign_sources
    return false
endif
```

on a system, accepting ident requests on port 113/TCP (i.e. port 113 is the default ident TCP port). Although no longer standard practice, some legacy servers (e.g. mail, FTP, and telnet servers) send an ident request to a client system in response to a connection request. If the ident service is running on the client an ident response is given. However, if ident is not an offered service most client systems simply respond with a TCP reset and the connection to the server still occurs normally. To not erroneously categorize such attempts triggered by legitimate local host activity as scans, any connection attempts to the ident port are simply ignored by the TRW algorithm.

The next check on line 9 of Figure 3.7 determines if the source IP address is contained in the `scan_sources` set. If it is, a boolean true is returned by the function (indicating this is scanning activity). This ensures that a source IP address already determined to be a scanner (i.e. having previously exceeded the TRW threshold) is not evaluated again so that additional alerts will not be raised as a result of previously detected scanning activity.

If the connection state is equal to `S0`, `REJ`, `OTH`, or `RSTO` this indicates that a failed connection attempt has occurred and the boolean variable `established` is set to false. In effect, this check on line 12 can be regarded as a *connection oracle* (TRW-oracle) that answers true or false to the question "Is the new connection attempt a success?" If `established` is false and the source IP and destination IP address pair is not in the `failed_locals` set, `flag` is set to 1 and the source and destination IP address pair is added to the `failed_locals` set. The `failed_locals` set is the list of unique hosts scanned by a scanner. Thus, subsequent failed connection attempts by a source IP address to the same destination IP address are ignored. The implication of ignoring repeated failed connection attempts from a source IP address to the same destination IP address is that only scanning activity to unique destination IP addresses will be considered when determining the source IP address's associated likelihood ratio for comparison with the TRW threshold.

If `established` is set to true, which means the connection has succeeded, then `flag` is set to -1 and the source and destination IP address pair is added to the `successful_locals` set (line 22 of Figure 3.7). Repeated successful connection attempts from a particular source IP address to the same destination IP address are ignored. This is done to ensure only successful connection attempts to unique destination IP addresses are considered when determining if a host has exhibited enough past successful connection attempts to be considered benign and thus exempt from further consideration as a scanner for a finite period of time (default one hour).

The `flag` variable determines how the `ratio` variable is defined, based on the values of `theta_one` and `theta_zero`. For instance, using the values found in Figure 3.6 for `theta_one` and `theta_zero`, the `ratio` variable would be updated to either 4, when `flag` is 1, or 0.25, when `flag` is -1 (see lines 28 and 30 of Figure 3.7).

The `updated_lambda` table holds a value for each source IP address being tracked. Each entry in the table contains a value equal to the product of the individual values of the `ratio` variable for each trial. Thus, the `updated_lambda` table provides an indication of each tracked source IP address's past connection behavior in terms of its connection successes and failures. Specifically, a failed connection attempt will cause the value of the associated source IP address's entry in `updated_lambda[ ]` to increase by a factor of 4 while a successful connection attempt will decrease its value by a factor of 4. If the `updated_lamba[ ]` for a given source IP address exceeds the threshold `eta_one`, the host at the source IP address is considered to be a scanner. The host's IP address is added to the `scan_sources` set, an alarm is raised, and a boolean value of true is returned. Conversely, if the value of `updated_lamba[ ]`is below the threshold `eta_zero` (e.g. .01 per line 13, Figure 3.6), the source IP address is considered to be benign. The host's IP address is added to the set `benign_sources` and a boolean value of false is returned. Based on these default settings, a host will be considered to be a scanner after a series of 4 failed connection attempts to unique hosts is observed (see equation (3.3)). Similarly, a host will be considered to be benign after a series of 4 successful connection attempts to unique hosts is observed (see equation (3.3)). If some combination of successful and failed connection attempts are observed by the host, additional connection attempt observations (the specific number will vary depending on the settings of the variables listed in Figure 3.6) will have to be made in order to determine if the host can be classified as a scanner or a benign system.

This specific implementation of TRW (taken from [9]) however, also takes into consideration factors other than the succeed/fail information provided by the four designated connection states of Table 3.1 in an attempt to classify a system as either scanning or benign. These factors include the following.

i. The use of time windows. The connection successes and failures for a remote system are tracked and assessed using a 30 minute (default value) time window.

ii. Previous scanning activity. Once a remote host has been identified as a scanning system, subsequent notifications of any additional scanning activity are suppressed for one day. Thus, a detected scanning system can generate a maximum of a single alert per day.

iii. Identified friendly remote hosts. If a local host initiates a connection with a remote host, subsequent connection attempts (even failed attempts) are ignored from the remote host when determining if the remote host is undertaking scanning activity. A possible exception to this occurs if the local host attempts to connect to a remote system using a protocol known to make a local host initiate a connection attempt to a remote host in response to receiving one (i.e. `triggered_outbound_services`).

iv. Identified benign remote hosts. If a remote host makes a number (based on a factor determined by the parameter settings previously discussed) of successful connection attempts, it is considered to be benign and exempt from further consideration as a scanner for a period of one day.

v. The number of unique hosts scanned by a remote host. Failed connection attempts from a remote host to a unique local host are added to the remote host's anomaly score for comparison against the TRW threshold. However, subsequent failed connection attempts from the remote host to the same local host are not considered during a predetermined time window (i.e. 30 minutes, see item i.).

Weaver et al. [70] developed a scan detection and suppression algorithm based on a simplification of TRW that makes it suitable for both hardware and software implementations. Similar to the original TRW algorithm [29], the success or failure of a connection attempt is only tracked for new IP pairs (i.e. a unique source and destination IP) with repeated connection failures between the same IP pair exempt from further evaluation. The technique uses approximate caches (i.e. caching approximate values instead of exact values for performance gains) to store the first occurrence of IP pairs associated with a failed connection attempt, connection attempts to new IP addresses, connection attempts to new ports at old addresses, and connection attempts to old ports at old addresses if the associated entry in the cache has been been timed out. One drawback with using an approximate cache is that a new entry in the cache may generate an insertion collision (i.e. a different IP pair may hash to the same index entry in the cache) and necessitate either an entry deletion or some form of entry combination. The authors chose to deal with collisions by combining the entries in such a way that allows false negatives but not false positives, thus making the technique suitable for automated response options. Any entry in the cache idle for more than ten minutes is automatically deleted. This technique can detect scanning worm infections with a sustained scanning rate of one scan per minute, within the first 10 scanning attempts. The authors also devised a method to implement this algorithm using cooperation among the worm detectors deployed within the network cells to rapidly contain worm outbreaks.

3.2 External Network Scanning Detection Strategies – Other

TAPS [63] is a connectionless portscan detection algorithm that can perform fast scan detection on high speed backbone networks. The detection of port scanning activity in large transit backbones is a challenging problem due to the speed, complexity (i.e. traffic diversity), and lack of configuration knowledge of the hosts that communicate within the respective links. The algorithm is based on a time-based access pattern sequential hypothesis testing algorithm, whose underlying premise is that a scanning host's connection behavior will exhibit a high connection ratio of unique IP address/port (number of destination IPs/number of ports) combinations that exceeds some predetermined threshold within a specified time period. Based on this ratio, it will be possible to determine (statistically) if a host is exhibiting scanning or benign behavior within a specific time *bin*. A test for the hypothesis of whether a host is benign or a scanner is then performed over a number of time bins. TAPs relies solely on counting the number of distinct IP address/port combinations a system has accessed within a given time bin. As no connection state information is required to perform this counting, it makes the algorithm ideal for detection of both UDP and TCP scanning activity within high speed links. TAPS is implemented in an architecture called CMON. CMON uses specialized hardware (i.e. DAG packet capture cards) to operate at backbone wire speed (e.g. 10 Gbps). The authors provide a good description and justifications for their proposed algorithm and the trade-offs they have made (i.e. Flajolet Martin distinct counters, number of hash functions required, implications of accuracy of the distinct IP addresses and ports recorded) to work at backbone speed. They further claim that in side-by-side comparisons with both Snort and Bro, TAPS exhibited the best performance in terms of detection and false positive and negative rates.

The basic idea of *exposure maps* was introduced in a position paper [73], and developed as an example of an attribution-free scanning detection technique. Analysis (which we justify in Section 5.1 of this thesis) indicates that it can detect both sophisticated and simple forms of network scanning activity.

Passive scanning techniques continuously monitor the hosts and services available in a network (see Section 2.1). *Extrusion detection* [7] refers to identification of unauthorized internal network activity by inspecting outbound network traffic; it is used to identify outgoing attack attempts from compromised internal systems and identify unauthorized network activity (e.g. removal of sensitive information from the network, evidence of internal attacks).

In contrast, *active* scanning[2] software (see Section 2.1), both open source and commercial, allows a security audit on a host or network [21, 67, 54]. Active scanning can be an integral part of a security audit to confirm that a host or network is in compliance with the network security policy. This activity however, can be costly in terms of human resources as it requires personnel to perform the required scanning

[2]Active scanning involves injecting packets into the network in order to elicit some observable response.

activity (i.e. configure and operate the software) and interpret the results. Furthermore, active scanning provides only a *snapshot* in time of the active hosts and services in a network. Any new hosts or services offered by the network will only be detected at the next scheduled active scanning session.

Leckie et al. [31] use probabilistic models of port connection information in order to detect scanning activity. These models require that each host in the local network be assigned a probability of how likely a given remote host will try and connect to it. In essence, the model assigns an access distribution probability for each host in the local network in terms of the number of remote hosts that have tried to make a connection to the specific local host. This technique relies on the premise that remote scanning hosts will access local hosts with equal probability during a scan. Thus, the distribution of scanning systems and benign system (i.e non-scanners) should exhibit a strong modality that will allow scanning systems to be identified by their connection patterns. This technique has some limitations. Specifically, if the local network contains a small number of busy systems (e.g. servers), the probability distribution for less busy systems (e.g. clients) may be characterized too low and introduce a high rate of false positives. To remedy this, the method used to assign probabilities to individual systems should take this scenario into account. Additionally, as with any technique that relies on a training period, any ongoing scanning activity present in the training dataset will skew the probability distribution of the model for the local network incorrectly base-lining some scanning activity as legitimate connection behavior.

Stealthy Portscan and Intrusion Correlation Engine (SPICE) [64] was a DARPA-sponsored project developed by Staniford et al. to detect port scanning activity. The technique is based on two components, a network anomaly detector (SPADE – Statistical Packet Anomaly Detection Engine) and a correlation engine. Port scanning activity is detected by assigning anomaly scores to incoming packets. Specifically, SPADE estimates the probability distribution of normal network traffic and assigns each packet an anomaly score based on an associated entropy measurement. This technique relies on the assumption that the more frequently a port/IP address combination is accessed, the less anomalous it is for packets to be sent to that particular port/IP address combination and thus a lower associated anomaly score should be assigned. The anomaly score is derived by taking the negative log of the probability of a packet being sent to a particular port/IP address combination. SPADE was implemented as a Snort [56] preprocessor. SPADE will alert on packets that exceed a predetermined anomaly score threshold. The correlation engine keeps the alerts generated by SPADE as events stored in main memory. Events are inserted into graphs where each node represents a packet and links between the nodes represent a connection. Links are assigned weights that indicate the strength of correlation between nodes. The more anomalous the score associated with the alert, the longer the event will be kept in the graph. The correlation engine then tries to link the events into groups of associated events that may indicate a scanning campaign.

Network *darkspace* is the unused IP addresses in a network and thus it should have no legitimate network activity directed to it; connection attempts to IP addresses that have no hosts assigned are considered anomalous. A number of commercial products (e.g. [36, 19]) make use of network darkspace to detect malicious network activity. A *darknet* is typically a large unused block of Internet-routable darkspace monitored for inbound packet activity. The larger the darknet, the better the darknet's ability to detect scans and attacks during an observation period [40, 38]. A related but subtly different approach by Harrop et al. [25] uses *greynets*, defined as regions of darknet address space that contain some active systems (i.e. some of the IP addresses in the darknet are assigned to active hosts). One of the motivations is that it is not possible for most enterprise network operators to have large regions of contiguous unused address space assigned to them. However, it would be useful to have some means to detect anomalous events if dark space was available on the network. Interspersing valid light (i.e. used) and dark (i.e. unused) addresses throughout a network will presumably make it difficult for malware to avoid targeting greynet addresses.

The observation of network service use, such as DNS, offers a means to detect anomalous network activity. Kruegel et al. [30] proposed the use of application specific knowledge of network services to enable detection of malicious content in individual packets. Their approach was to use statistical anomaly detection to detect R2L attacks targeted at essential network services. Anomaly scores for specific packets are based on deviations from expected values in a predetermined profile. Once a threshold is exceeded, an alarm is generated. They based their experimental analysis on a prototype that processed both HTTP and DNS network traffic.

Snort is an open source IDS that has scanning detection capability [56] through the use of the Snort preprocessor sfPortscan [55]. Snort's preprocessors work by implementing a plug-in model. Specifically, any detection algorithm that requires some form of packet decoding or transformation can be implemented as a preprocessor making the associated network traffic easier for Snort to handle. Preprocessors are flexible and provide a variety of functions including alerting, classifying, or dropping packets before they are passed to Snort's detection engine. sfPortscan provides the capability to detect TCP, UDP, and ICMP scanning; its sensitivity is set using the *sense level* parameter (low, medium, or high). Types of scans detected by Snort include: 1) *portscans* (single host scans multiple ports on a single host); 2) *distributed portscans* (multiple hosts scan multiple ports on a single host); and 3) *portsweeps* (single host scans a single port on multiple hosts). The sfPortscan preprocessor detects scans by counting RST packets from each perceived target during a predetermined timeout interval [32]. Before declaring a scan, 5 events (i.e. RST packets) are required from a given target within a window. The sliding timeout window varies from 60 to 600 seconds by sensitivity level; at the highest level, an alert will be generated if the 5 events are observed within 600 seconds.

3.3 Internal Network Scanning Detection Strategies

Ganger et al. [23] present a software architecture to enable self-securing network interfaces to examine packets as they move between network links and host software, detecting and potentially blocking malicious activity. This host-based approach includes a detection technique that enables detection of scanning worm propagation. The technique involves shadowing a host's DNS table and checking the IP address of each new connection against it. The basic premise of this approach is that its abnormal for a host to make a large number connection attempts without DNS activity.

Williamson [76] devised a method to limit or throttle the rate of malicious mobile code by determining the rate at which a host is trying to connect to new IP addresses.

This algorithm is based on the assumption that when a human interacts with a host during normal activities (e.g. web surfing, email) new connections to unique systems will be at a low rate whereas a worm infected system that requires no human interaction to spread will have a high rate of new connection requests to unique hosts. Thus, this technique relies on the hypothesis that during normal application usage, a stable rate of new connections is initiated by a non-malicious host. A host trying to make rapid connections to multiple unique IP addresses could indicate some form of malicious scanning activity. Hosts that exhibit this type of behavior should be limited or throttled to restrict the rate of new connections. The rationale for throttling hosts with high connection rates to unique IP addresses is that targeting the propagation method (i.e. reducing the scanning rate) of a scanning worm infected system you limit the rate of infection. Each host of interest has a *working set* of hosts that models its past connection behavior. Outgoing connections to any host within the working set experience no delays. Connections to hosts outside the working set are placed in a delay queue until the queue is full at which point new connections are simply dropped. The queue makes use of of a least recently used (LRU) eviction strategy to drop connections to IP addresses that have been accessed least recently. In this way, new connections to random IP addresses are prime candidates for eviction while connections to previously contacted hosts are more likely to be allowed. The paper recommends a working set of five hosts and a queue length of 100 IP addresses. The small working set size ensures that the scheme is somewhat restrictive in allowing rapid connections to new unique hosts. The suggested delay queue length however, allows connections of up to 100 additional hosts with only a small delay and without any of the new connections being dropped.

Wong et al. [77] performed an empirical analysis of rate limiting mechanisms. The techniques they analyzed were designed to mitigate the propagation of scanning worms by slowing down the rate at which infected hosts can initiate connections to new victims while limiting the impact on legitimate systems. With all the rate limiting schemes, false positives impact legitimate hosts while false negatives will allow worm propagation to occur. The authors examined three rate limiting techniques, Williamson's IP throttling technique [76], a failed connection rate limiting technique by Chen et al. [11], and a credit-based rate limiting technique by Schechter et al. [58]. As well, the authors proposed a new rate limiting technique based on DNS. Williamson's technique relies on limiting the connection rate of hosts based on the number of connections to unique hosts while both Chen and Schechter's techniques rate limit hosts based on the number of failed connection attempts. The DNS-based rate limiting technique is based on the observation the worms make connection attempts without first making a DNS request. In this scheme, connection requests to systems that have been located using a DNS request are permitted without delay. Connection requests to systems in which a preceding DNS request has not been observed are delayed using a cascading bucket scheme. Specifically, connections that have had no prior DNS requests associated with them are added to a bucket (i.e. as an entry in a bucket which is a finite queue) and delayed for a specific time interval. Bucket sizes are finite and once a bucket is full, any new connection requests that need to be added to a bucket are stored in subsequent buckets; thus one bucket cascades into another. The greater the number of connection requests that need to be stored in the buckets, the greater the delay imposed on new entries. The number of buckets is also finite and once all buckets are full, any new connections are simply dropped. The technique can be implemented at both the host and network level. A comparison of all techniques revealed that the DNS-based technique had the lowest rate (i.e. below 1%) of both false positives and negatives.

Dagon et al. [14] use a modified collection of honeypots to detect scanning activity in a local network using a scheme called Honeystat. A number of services are run virtually on a host that monitor main memory, hard disk and the network for the most common exploited software vulnerabilities. The type of honeypots used for Honeystat are both high interaction and multi-homed (to emulate a larger IP address space so that the likelihood of an exploit attempt is greater). All relevant data associated with a generated alert is captured for secondary analysis. This includes operating system type, application patch versions, network stack state, main memory events, and network packets associated with the exploitation attempt. Unlike traditional honeypots, Honeystat nodes are efficient script-driven systems that monitor large ranges of IP addresses. The additional benefit of this approach is that it can provide more information than a traditional honeypot including identification of binary signatures and specific attack vectors.

A technique developed by Sharif et al. [24] uses a two-phase local scanning worm detection algorithm call DSC (Destination-Source Correlation) that takes into consideration both infection patterns and scanning patterns of worm-infected systems. The technique is deployed on distributed sensors within the local network and can incorporate local victim information (i.e. the actual scanning strategies the infected system employs) to enable practical real-time response to stop internal network worm propagation.

3.4 Summary

To effectively minimize the success of automated attacks as a result of scanning activity, detection must occur regardless of whether the scanning activity originates from within or outside of the network boundary. Although some progress has been made to develop techniques to detect localized (i.e. internal) scanning activity, there is a dearth of published research in this area. It can be argued that the most serious threats to a network occur from compromised systems within the network due to a typically higher level of trust given to internal systems and the tendency for network operators to install countermeasures primarily at the network boundary. New research in this area is important to address a critical gap in the available detection capabilities for internal network scanning activity. We pursue this in chapters 6 and 7 in this thesis.

More progress has been made in the development of network scanning detection techniques that attempt to identify external network scanning activity. However, most of these techniques rely on the identification and correlation of external connection events to detect scans. This introduces a serious limitation in that with mechanisms known to date, a significant amount of system state (e.g. main memory, network topology information, secondary storage) needs to be maintained by the monitoring system in order to perform effectively. These techniques to date have employed strategies to minimize the amount of state information maintained by the monitoring system by using such methods as reducing the detection time window in which connection events are tracked or using timeouts to accommodate network traffic fluctuations. These *graceful retreats* can cause both false negatives and positives. In contrast, in chapters 4 and 5, we describe a technique to detect external scanning activity that requires only a minimal amount of state information that does not fluctuate in proportion with increases in detected external scanning activity.

Chapter 4

Exposure Maps: Approach and Evaluation Methodology

This chapter proposes a new R2L network scanning detection technique called exposure maps/ darkports. Specifically, we describe how exposure maps can be used in the following security applications: (1) scanning detection, (2) automated response, (3) host discovery and asset classification, and (4) network service enumeration. Our discussion will include design and implementation details of our techniques using a suite of Bro policies. Finally, we will describe the three network datasets used to evaluate and test our techniques, the results of which are discussed in chapter 5.

4.1 Exposure Maps: Approach

Exposure maps passively identify the services which have been confirmed (through an observed response during a training period) as being offered by the hosts of a given network. TCP packets with the SYN flag set start the three-way connection handshake. When a connection request is sent to a specific destination IP address/port, if a service is bound to that port and the port is listening (open), the target host response is a packet with SYN ACK flag set, to start a session. Listening services, because they respond to connection attempts or incoming packets, leak information to scanners; they typically correspond to the active ports in a network and can be tracked in terms of what we define below as the HEM and the NEM. Once verified as permitted activity, the HEMs and NEM define the authorized access to individual hosts and the network.

Each UDP datagram can be regarded as a discrete event and a potential new communication between hosts. As UDP is connectionless, other measures must be taken to identify and track communications streams between host pairs exchanging UDP datagrams. For instance, two hosts observed exchanging UDP datagrams will be considered by convention, as participating in a *session*, with the host that initiates the exchange considered the client. A host that responds (after receiving an initial UDP datagram) by sending back a UDP datagram will be regarded as a server and its corresponding UDP source port is regarded as open. A HEM for the UDP protocol, associated with a fixed IP address, is the set of ports observed responding to an initial incoming UDP datagram that signifies the start of a UDP datagram exchange.

More specifically, a *host exposure map* (HEM), associated with a fixed IP address (host), is the set of ports observed over a predefined period, responding to external connection attempts. For each active host i in the network, HEM_i is a set of elements each of which begins with the IP address of i, followed by a port number j; there is such an element for each $port_j$ that has responded to a connection attempt within a predefined period. In symbols, we can abbreviate this as $HEM_i = \Big\{IP_i : port_j \;\Big|\; port_j \textit{ was observed responding}\Big\}$.

The HEM is the externally visible interface of a host and can be considered to represent information leakage from the host that may reveal characteristics that can be used to exploit it. Subsequent to the training period, as additional ports respond to external connection attempts, by definition the HEM is augmented by these ports.

The *network exposure map* (NEM) is defined as the collection of HEMs in a given network N at any given point in time. The NEM defines how we expect the network to respond to external connection attempts. In symbols, $NEM_N = \bigcup_{i \in N} HEM_i$. We will often drop the subscript N in NEM_N when the target network is implied by context. This also allows the natural definition of NEM_S for any subnetwork $S \subset N$, i.e. where S is a subset of the *populated* IP addresses in N. In this context, we define the term populated IP address as an active host (i.e. not network darkspace).

In an implementation, once the NEM has been built, the individual HEMs that comprise it can be checked for compliance with the network security policy. A NEM that complies with the network security policy is called a *vetted NEM*. We assume that any service (IP address/port pair) not compliant with the network security policy will, once detected, either be shutdown, or implicitly becomes part of the security policy. Thus, movement from a NEM to a vetted NEM is always possible where a defined and verifiable network security policy exists. Figure 4.1 shows a NEM composed of six HEMs.

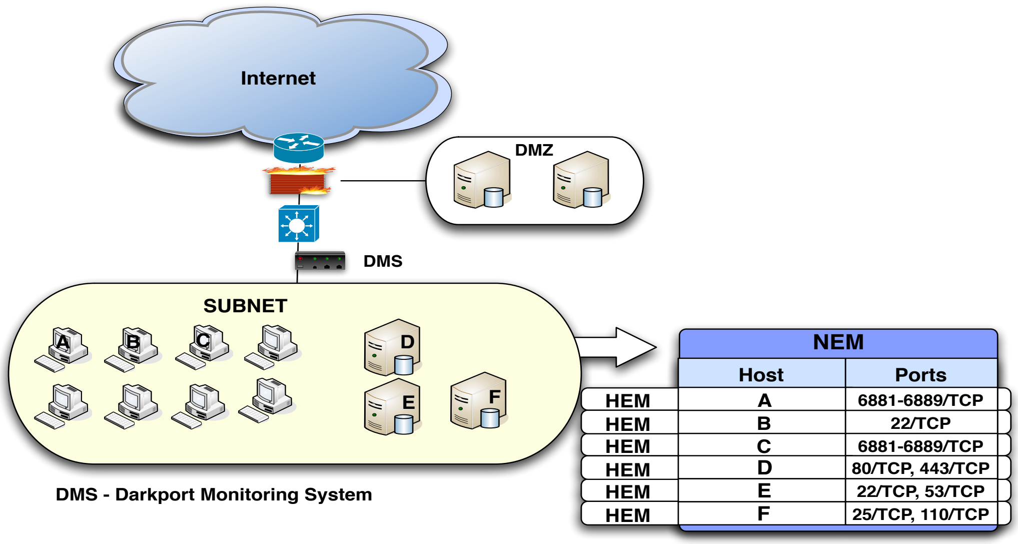
In some network environments, the *defined* network security policy may be one that is, by design, very permissive. For instance, specific rules concerning allowed network activity may be defined for the systems within a few secure enclaves in the network while other subnets and/or systems in the network will have little or no restrictions placed on their associated network activity. In this instance, NEM construction and vetting could be limited to only the systems that comprise secure enclaves.

Conversely, some network environments may simply be too diverse, distributed, or transient (e.g. permit mobile devices, guest users) to enforce and apply a single uniform network security policy. In these cases a NEM can still be constructed but not vetted against a network security policy. Although such a NEM is referred to as an *unvetted NEM*, it is, by lack of a meaningful security policy, analogous to a vetted NEM. Specifically, due to the permissive nature of the network, all services are simply allowed and deemed to be compliant with standard operating procedures. One important difference between a vetted and an unvetted NEM is that an unvetted NEM does not require a training period during its construction and maintenance (see discussion on Exposure Map Construction and Maintenance below).

We define the *darkports* on a given (populated) host as those ports that have not been observed offering any services (during the training period in the case of a vetted NEM), and thus are not expected to accept external connection requests (TCP) or UDP datagrams.[3] The set of darkports for a host is the complement of its HEM. The set of darkports for a network is the union of the darkports on all its populated hosts. For example, a host with a HEM of only three TCP ports 22, 80, and 443 would have $2^{16} - 3$ TCP darkports i.e., all TCP ports excluding these three. If a darkport responds to an external connection attempt, it becomes a *trans-darkport* (transitioned darkport). In the case of a vetted NEM, this occurs either when a host offers a new service (whether authorized or rogue), or a connection is made to a service that was not accessed during the training period. Either event causes the HEM to expand, and by definition the NEM expands and will differ from the vetted NEM. Once a

[3]Although a connection attempt to any port at a darkspace IP address (no hosts assigned) will not accept a TCP connection attempt or UDP datagram, we restrict the term darkport to unused ports on a populated host address.

trans-darkport is detected, service on this port can be checked against the network security policy so that the vetted NEM can be updated or any unauthorized service can be stopped. In the case of an unvetted NEM, the NEM automatically expands as new services are offered (i.e. no checks are made against a known security policy). Although a new service detected in the network will be automatically entered into an unvetted NEM, this does not preclude checking the NEM on an ad hoc basis to determine if any known malicious network services (e.g. known backdoor trojan ports) are being offered by hosts on the network (see discussion in Section 4.1.3)



NEM

	Host	Ports
HEM	A	6881-6889/TCP
HEM	B	22/TCP
HEM	C	6881-6889/TCP
HEM	D	80/TCP, 443/TCP
HEM	E	22/TCP, 53/TCP
HEM	F	25/TCP, 110/TCP

Figure 4.1: Example NEM.

Exposure Map Construction and Maintenance. In summary, exposure maps are created by passively observing a target network's responses to incoming connection attempts (both legitimate connections and scanning attempts) or UDP datagrams with or without the benefit of a training period (respectively as part of a vetted or unvetted NEM respectively). Every time a host responds to an external TCP connection attempt or incoming UDP datagram, the IP address and port of the host offering the service is recorded. While the construction and maintenance of a vetted and unvetted NEM is similar, there is potentially a fundamental difference in the way that trans-darkports will be interpreted by the network operator.

In the case of a vetted NEM, each host in the network during the training period will reveal services that it offers; the corresponding ports are recorded in its HEM. After the training period, the vetted NEM can be used to identify all the active hosts on the network by their representative HEMs. Thereafter during ordinary network operation, passive observation of network packets continues, and for each connection attempt (i.e. each TCP SYN packet or incoming UDP datagram) compliance with the vetted NEM is tested in real-time. If the services offered by a host expand beyond the vetted NEM, an alert is generated to provide notification that trans-darkports have been detected; this indicates to the network operator that either the vetted NEM needs to be updated, or some form of unauthorized activity is occurring.

In general, an important consideration for any technique that requires a training period is that any existing malicious activity (e.g. unauthorized services) may become part of the baseline. In our particular case, a HEM can be verified against an existing network security policy to detect any unauthorized service offerings by the host. The required length of the training period will vary with each network environment depending on a number of factors including number of active hosts, network security policy, permitted user applications, and frequency of service usage; see Section 5.2 for further discussion.

An unvetted NEM will typically be used in a very dynamic network or one where compliance with a network security policy is either not practical or not possible. Thus, we expect that during ordinary network operation new hosts and services will be revealed and the unvetted NEM will expand in near real-time. An alert can be generated (or suppressed as a configuration option) to notify that a new network service and/or host is active as determined by the emergence of a trans-darkport.

A number of possibilities exist for automatically removing stale (i.e. inactive or unresponsive hosts and/or services) entries in a (vetted or unvetted) NEM. One possible method is to keep track of the last access time of each NEM entry when processing incoming connection attempts during construction and ongoing maintenance (as described above). Entries that have not been accessed within a certain period of time can be automatically deleted or an alert can be generated to notify that the NEM entry is a viable candidate for deletion (see Section 4.1.5).

4.1.1 Use of Exposure Maps for Scanning Detection

To use exposure maps for scanning detection, first a vetted NEM is constructed as previously described. A connection attempt to any port-IP combination not present in the vetted NEM (i.e. a darkport or darkspace) is defined as a *darkport connection attempt* (DCA). A DCA represents a connection attempt (often resulting in a failed connection) to a host or service not offered in the network that is a typical indication that some form of scanning activity is occurring. The 5-tuple (source IP, destination IP, destination port, protocol, timestamp) of any DCA is recorded to secondary storage (hard disk) in a DCA activity log file for further analysis to determine the type and scope of scanning activity. This approach requires only that the NEM information be maintained in system detection state (not the darkports or external connection requests), thus allowing detection of all the connection attempts (i.e. DCAs) that constitute even very slow or distributed scans, using only a small amount of main memory. In contrast to most scanning detection techniques that rely on the identification and correlation of external connection events to detect scans, we thus do not require strategies like reducing the detection time window in which connection events are tracked or timeouts, to accommodate network traffic fluctuations. Connection attempts to hosts within the NEM to port 113/TCP (i.e. ident) and port 79/UDP (i.e. finger) are ignored to eliminate a possible source of false positives (see `triggered_outbound_services` discussion in Section 3.1).

Unlike most attribution-based scanning detection techniques, our scanning detection approach does not rely on identification of the scanning source to detect scans against a network. Thus, it can detect certain classes (see Section 5.1.2) of sophisticated scanning techniques that make determining the root cause of the scanning activity impractical. However, this approach does not preclude the use of some form of attribution *post* scan detection. Scanning worm propagation and auto-rooters are two prevalent examples of scanning activities where immediately denying the scanning source access to the network is both relevant and important. In these cases, a successful scan (i.e. one triggering a response from a host) typically leads to an immediate attack from the scanning systems. Other post scan detection activities may include the use of heuristics to classify DCAs into their respective scanning campaigns. An example of such a heuristic is given in Section 5.1.2 to identify and correlate DCAs that comprise a distributed scan. Alsaleh et al. [3] have successfully used the output of the exposure maps technique as a *filter* to select subsets of potentially malicious network traffic in order to detect sophisticated scanning activity (i.e. distributed scans) through visualization.

Analogous to a vetted NEM, any connection attempt to any port-IP combination not present in an unvetted NEM is considered a DCA and recorded as such. However, an unvetted NEM presents us with the following boot-strapping issue. Intuitively, we should expect the frequency of trans-darkport occurrences for an unvetted NEM to be higher than that of a vetted NEM, assuming the unvetted NEM is used due to the dynamic nature of the hosts or services offered. Therefore, we expect not all new connection attempts will match an unvetted NEM entry, and a successful connection to an unknown service (i.e. unknown to the unvetted NEM) may still occur. Such activity is classified as trans-darkport activity and results in the unvetted NEM being automatically updated. Unfortunately, the initial NEM-membership test of the destination IP and port/protocol of the new connection attempt, that is performed to detect a possible scan, will have already occurred and failed, resulting in a DCA being recorded. To account for this spuriously recorded DCA, a notification is generated that a DCA has been recorded as a result of trans-darkport activity and should be deleted (recall that DCAs are written to disk for subsequent processing). In practice, the vast majority of legitimate connection attempts (with the exception of the first observed connection attempt to a particular service that causes trans-darkport activity) are quickly checked against the unvetted NEM and discarded as not being potential scanning activity.

A vetted NEM has the benefit of a training period to determine the composition of a NEM and provides a priori knowledge of permitted connections. Although new service/host offerings for a vetted NEM are also considered trans-darkport activity, we expect such activity to be infrequent (otherwise an unvetted NEM should be used) and results in an alert to the network operator to confirm the newly detected service being offered is compliant before entry into the NEM.

To fully scan all the TCP services on a network (scanning of UDP services is analogous) of n hosts, a scanning tool would need to scan E $= n * 2^{16}$ ports. For instance, in a Class C or /24 network (254 hosts excluding broadcast addresses), $\approx 2^{24}$ unique TCP port/host pairs could be scanned. In practice, often only a subset of available ports is scanned, as attackers try to locate well-known services in the reserved port range (i.e. 0-1023) or backdoor trojan ports listening on ephemeral ports. Let A be the actual number of services scanned in a network, i.e. the number of unique IP/port combinations of all the detected scans. Within A, each scanning attempt can result in one of three possible outcomes: (1) a probe directed against a darkspace address, (2) a probe against a darkport (note: such a host has a HEM), or (3) a probe sent to a host on an active port (an entry in the NEM). Figure 4.2 shows the general relationship between the potential service ports scanned (E), actual service ports scanned (A), darkspace scanned, darkports scanned, and the NEM for a network.

4.1.2 Automated Response using Exposure Maps

Due to the dynamic nature of an unvetted NEM and its associated bootstrapping issue (see discussion in Section 4.1.1) it would be inadvisable to use it for automated response capability. Thus, we consider an automated response capability for network environments that employ vetted NEMs.

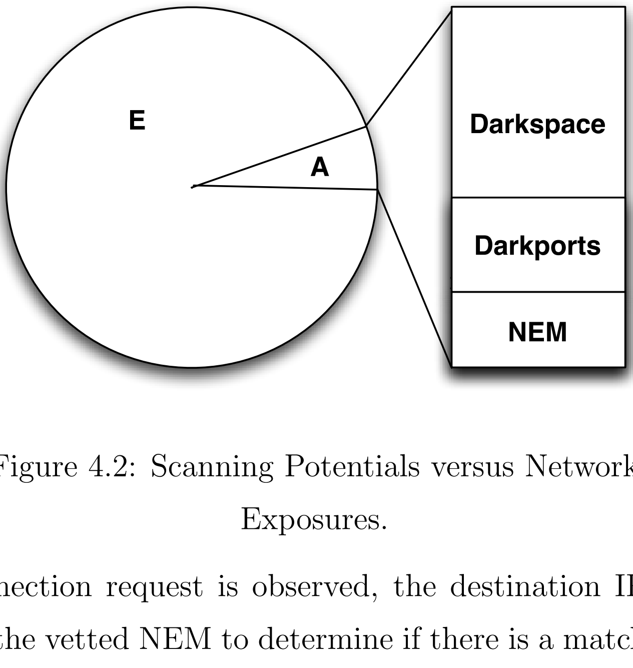


Figure 4.2: Scanning Potentials versus Network Exposures.

Exposure maps can be used in an automated response application as follows. When a new connection request is observed, the destination IP address and port are compared with the vetted NEM to determine if there is a match (see Figure 4.3). If there is no match to an entry in the NEM, the connection is considered a possible scan attempt and the source IP address is added as an element in a *scanners* list (implemented e.g., using a hash table). The 5-tuple that characterizes the connection attempt is then recorded as a DCA in the DCA activity log file.

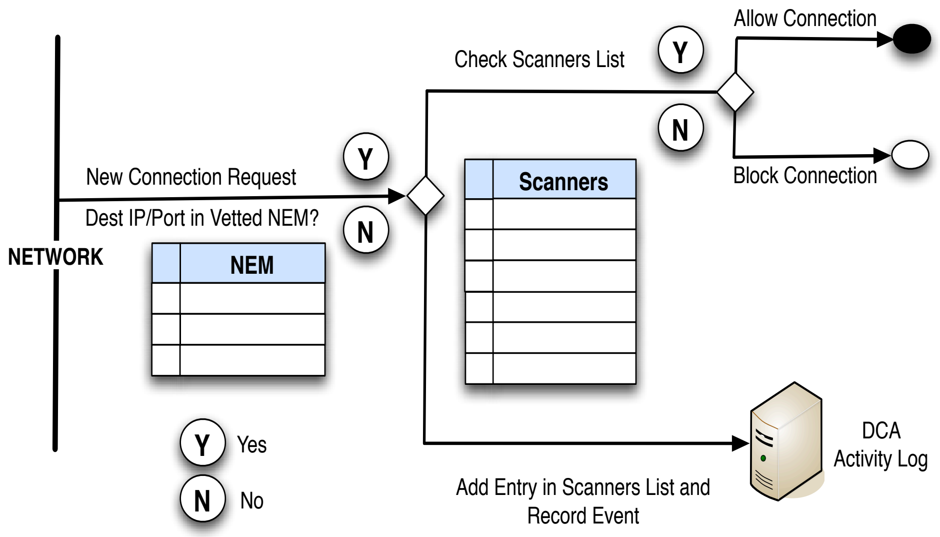


Figure 4.3: Exposure Maps Automated Response Logic.

On the other hand if there is a match, the source IP is checked against the scanners list. If the source IP address matches an entry in the list, the 5-tuple that characterizes this connection attempt is recorded and connection should be dropped as this entity has possibly undertaken reconnaissance activity against the network. Our implementation is passive and only produces alerts that could enable some form of containment (e.g. ACL change), but does not actually do the latter; one option would be to integrate this application on a network device capable of performing containment such as a firewall. If the source IP address does not match an entry in the scanners list, the connection is permitted; the entity has no previous history of scanning activity and is connecting to a valid service offered by the network.

The vetted NEM provides context to determine if an incoming connection request is part of a scanning campaign and whether it will likely elicit a response. This information provides us with the precision to limit containment to (e.g., automatically block) only those scanning systems targeting services offered by the network. Containment could alternately be performed using a number of network devices including firewalls, routers, or intrusion prevention systems using current scanning detection techniques. However, given the prevalence of scanning activity, frequent dynamic updates to these core network devices would be required in order to stop attacks in real-time, and would pose a number of challenges. For instance, Bobyshev et al. [8] have shown that the size of access control lists (ACLs) and the frequency of dynamic updates can significantly impact router CPU utilization and forwarding capabilities. Furthermore, the addition of multiple blocking rules may make ACLs and configuration files cumbersome and hard to vet by network personnel. In fact, frequent configuration changes to these network devices may actually decrease the overall security posture of the network over time [78]. The proposed technique allows a precise active response option to be taken exclusively against the most critical known threats to the network; namely, those scanning systems targeting services offered by the network. Scanning systems trying to access services not offered by the network are noted (i.e. in the DCA activity log and the scanners list) but no action is needed or taken to block the connection.

Our analysis on a four-week network data set reveals a majority of scanning attempts directed against services not offered by the network (see Section 5.1.3). In the instances when the scanning was directed against a service offered by the network, an attack almost always followed (see Table 5.10). Thus, our approach can significantly reduce the frequency and number of updates to the ACLs of network security devices while providing a measured and robust security response to real-time threats.

4.1.3 Exposure Profiles: Host Discovery and Asset Classification

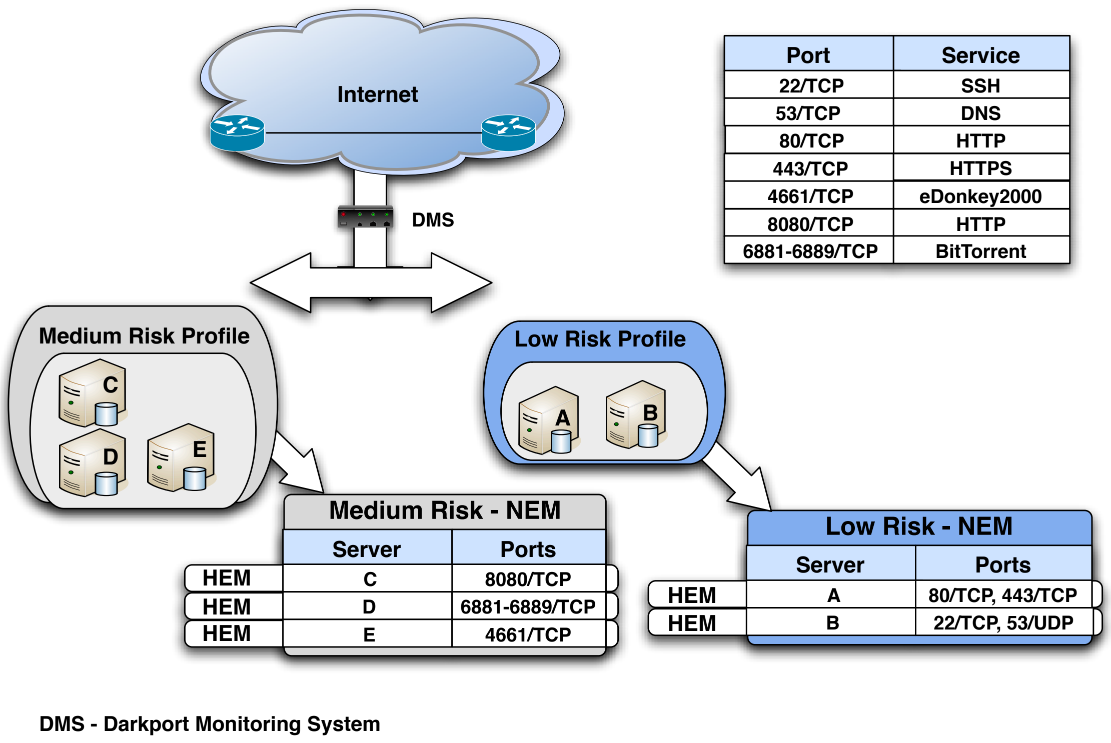
In large network environments, it may be useful to discern the number and types of systems within the network which offer services to external entities, and logically group them together. Exposure maps provide a mechanism to identify and group hosts that offer similar services into the same *exposure profiles*. As an example, the following three exposure profiles could be generated based on the perceived risk to the network:

i. *Low Risk*: web, DNS, mail, printing, network management.

ii. *Medium Risk*: open proxies, P2P services.

iii. *High Risk*: known worms, known trojan backdoors.

In general, the exposure profiles used would vary greatly in terms of the number and types of services in each, depending on the specific network. In our example, the low risk profile includes only well-known *traditional* services offered by the host. The medium risk profile indicates hosts that offer non-malicious but potentially risky services. The high risk profile denotes those systems that offer a service on a port that has known malicious activity associated to it. Logically grouping hosts by the contents of their HEMs provides a means to rapidly apply some action to a collection of similar systems if required (e.g. deny network access to hosts in the high risk profile to limit potential malicious activity). Furthermore, a change to a host's darkports may move it from one profile to another and necessitate some real-time action be taken on that specific host. Exposure profiles can be constructed using both vetted and unvetted NEMs. Figure 4.4 shows an example of two exposure profiles (low and medium risk) for a small network.



Port	Service
22/TCP	SSH
53/TCP	DNS
80/TCP	HTTP
443/TCP	HTTPS
4661/TCP	eDonkey2000
8080/TCP	HTTP
6881-6889/TCP	BitTorrent

Medium Risk - NEM

	Server	Ports
HEM	C	8080/TCP
HEM	D	6881-6889/TCP
HEM	E	4661/TCP

Low Risk - NEM

	Server	Ports
HEM	A	80/TCP, 443/TCP
HEM	B	22/TCP, 53/UDP

Figure 4.4: Exposure Profiles.

Exposure profiles are useful for other applications. An ISP could use exposure profiles in response to global cyber events (e.g. worm outbreak, new exploit, botnet DDoS attack) creating an applicable profile to identify hosts that are at risk or exhibiting signs of a successful compromise. Accordingly, to ensure the efficient use of resources, different monitoring thresholds and security applications could be applied to the subset of hosts within the network based on their exposure profile. For instance, hosts that belong to a server farm exposure profile (e.g. hosts that offer HTTP services in a network enclave) might be afforded a different type or greater level of monitoring than hosts with other exposure profiles. At the most basic level, exposure profiles could be used to simply differentiate between clients and servers in a network, or for network operators simply as a method to discover and baseline the services offered in their network.

4.1.4 Determining the Success of Network Service Enumeration as a Result of Scanning Campaigns

The overall goal of network scanning can vary among scanning entities, although it is typically to gain information about the network and the hosts that compose it, for an immediate (e.g. worms) or future (e.g. following some form of analysis on the scanning results) exploitation attempt. Attackers have a variety of scanning strategies at their disposal (see Section 2.2). In the case of more patient adversaries, considerable time may be taken to carefully probe the network to both avoid detection and gather as much information as possible before the formulation of a suitable attack vector. For instance, an attacker may have acquired an exploit that allows a buffer overflow to be executed on an HTTP server that gives them remote root access and the ability to execute arbitrary code. Buffer overflows however, are typically application-specific and will only work on a particular version of software running on a specific underlying hardware platform. In this example, it is an attacker's best interest to determine not only that an HTTP server is present in the network, but also the type, version, and patch-level of the application software it is currently running. Most servers routinely divulge this type of information during normal operation (see Section 2.1).

It would be useful for a network operator to utilize such data to know what application information about network hosts is being divulged to remote systems. This would allow a *catalogue* of the host application information sent from the network to be enumerated, recorded, and reported in near-real time. This catalogue could be used to confirm that the servers in the network are running the proper versions of approved applications in compliance with the network security policy. Secondly, it could provide a mechanism to determine what network (i.e. IP address, port, protocol) and application (e.g. banner, server strings) information was returned to an attacker during a scanning campaign.

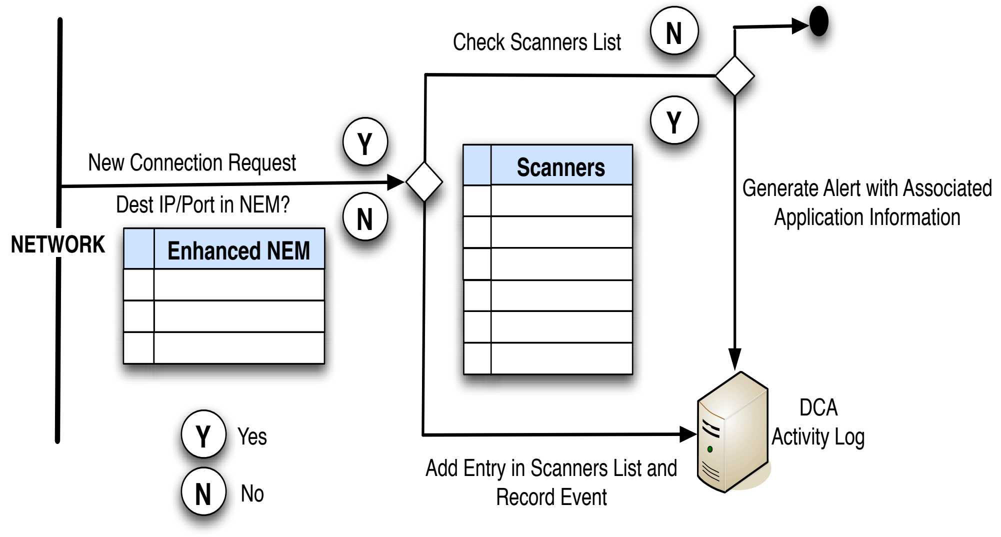
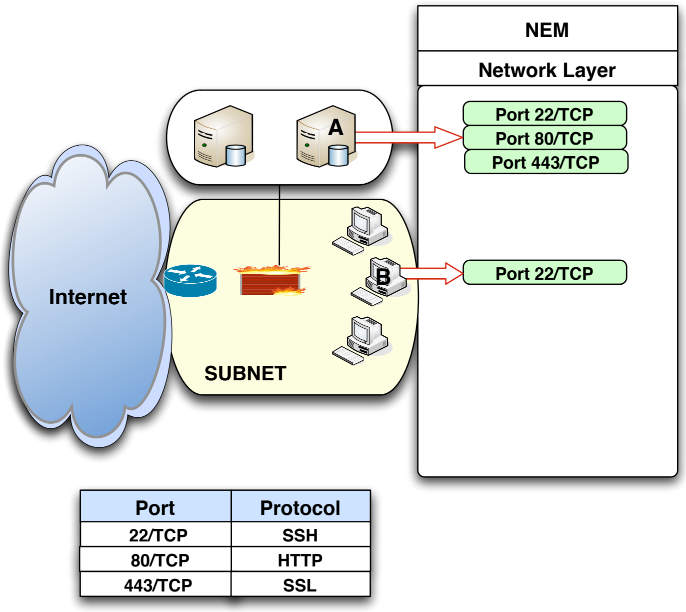


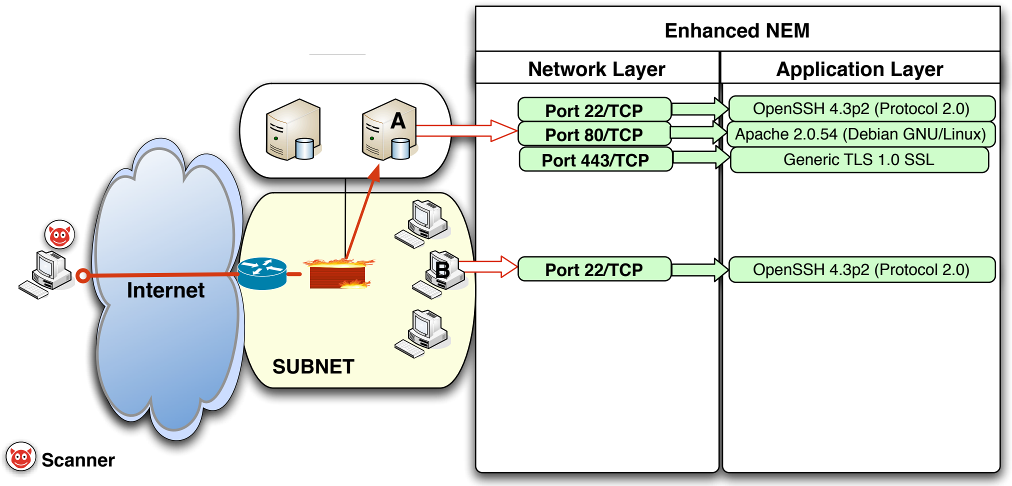
Figure 4.5: Enhanced NEM - RAA Logic.

Enhanced NEM Construction and Maintenance. We define the *enhanced* NEM as a basic NEM augmented to incorporate application information (e.g. banners) as well as network information (e.g. IP address, port number/protocol). An enhanced NEM may be vetted or unvetted. The vetting of an enhanced NEM would require confirming that hosts in the network were offering both the proper services and running approved applications in accordance with the network security policy. An enhanced NEM is created in the same manner as a basic NEM, by passively observing a target network's responses to incoming connection attempts. However, every time a host responds to an external TCP connection attempt or incoming UDP datagram, any transmitted application banners are recorded and associated with the IP address



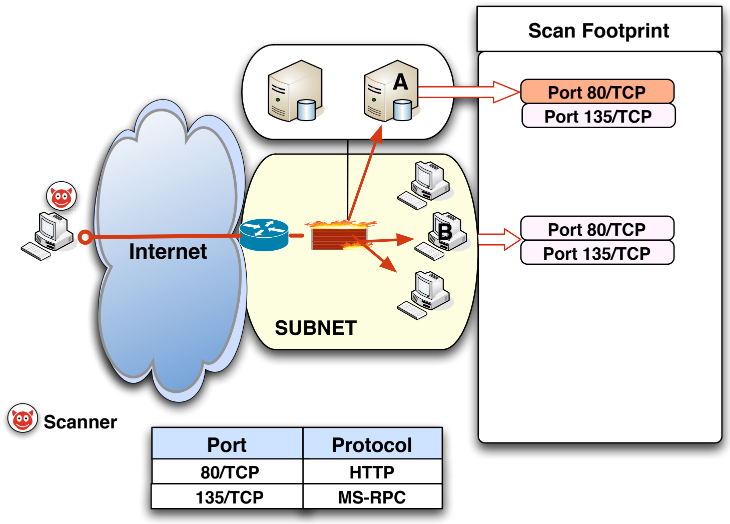
Port	Protocol
22/TCP	SSH
80/TCP	HTTP
443/TCP	SSL

(a) Basic NEM.

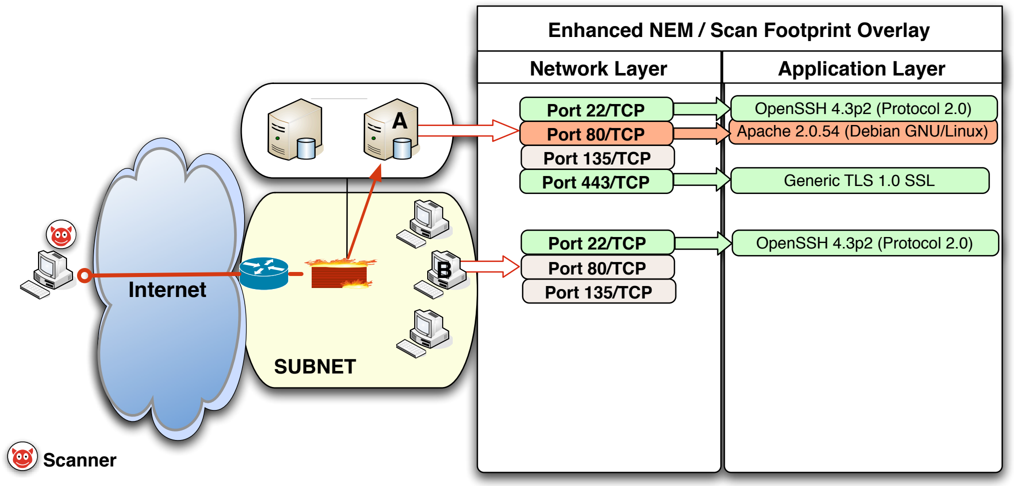


(b) Enhanced NEM.

Figure 4.6: Basic and Enhanced NEM.



(a) Scanning Campaign Overview.



(b) Enhanced NEM Overlaid with Scan Footprint from a Scanning Campaign.

Figure 4.7: Example of an RAA.

and port/protocol of the host offering the service. For example, an entry in the enhanced NEM could look like `[192.168.1.1:80/TCP:Apache 2.0.54 (Debian GNU/ Linux1)]`. Recall that a HEM, associated with a fixed IP address, is the set of active ports responding to a connection attempt within a predefined period (see Section 4.1). Each one of these active ports responds (typically) as a result of offering a service.

Reconnaissance Activity Assessment (RAA). An enhanced NEM provides an expectation of how the network will respond to external connection attempts at both the network (i.e. TCP/IP) and application layer. An enhanced NEM can be used to determine the network services enumerated due to scanning activity, known as an RAA, as now explained (see Figure 4.5).

When a connection request is observed, the destination IP address and port are compared with the NEM (vetted or unvetted) to determine if an entry is a match. If there is no match to an entry in the NEM, the connection is considered a DCA; the source IP address is added as an element in a scanners list and the 5-tuple that characterizes the connection attempt is recorded as a DCA in the DCA activity log file.

If there is a match, the source IP is checked against the scanners list. If the source IP address matches an entry in the list, the 5-tuple that characterizes this connection attempt is recorded as usual as a DCA. Additionally, any application information within the enhanced NEM associated with the destination IP and port/protocol is assumed to have been sent to the remote host initiating the connection. An alert is generated that notifies what application information (i.e. string that contains the application banner) has been sent to the remote host as a result of this connection. The collection of all such alerts generated by a specific remote host provides a mechanism to group and therefore determine what network services were enumerated as a direct result of a specific scanning campaign.

An RAA reveals the relative success of a specific scanning campaign in terms of the network services that were revealed (i.e. enumerated) verses the amount of scanning activity undertaken on the network. More specifically, an *RAA* is defined as the network service information[4] gained by an adversary during reconnaissance

[4]Here, network service information includes network or host topology, open ports, offered services, OS system versions, application software versions, and configuration specifications.

activity as a direct result of executing a specific scanning campaign.

The *overall network service information revealed by the network* is defined as network service information divulged by the hosts on a network as a direct result of external connection attempts (both legitimate connection attempts and network scanning activities). The respective entries in an enhanced NEM in its entirety represents a comprehensive corpus of information leakage from the network to remote hosts.

The following example illustrates how an enhanced NEM is used to provide an RAA. Figure 4.6(a) shows a NEM composed of two HEMs that offer services on a total of four ports. Figure 4.6(b) shows the same network using an enhanced NEM that reveals not only the open ports/protocols but the applications running on these ports based on passive observation of application banners sent to remote systems.

Consider the scenario of an adversary with a scanning footprint depicted in Figure 4.7(a). Hosts A and B were scanned on ports 80/TCP and 135/TCP respectively. In this example, only host A responded to the scan on a single port (i.e. port 80/TCP in red, which corresponds to a NEM entry) and the attacker was sent an HTTP Server string. The rest of the scans (to port 80/TCP on host B as well as all scans to port 135/TCP) are shown in grey as no response was given as a result of the scanning activity. An examination of the scan footprint (i.e. the set of IP addresses/port combinations an attacker is interesting in characterizing) overlaid with the enhanced NEM (see Figure 4.7(b)), reveals that port 80/TCP is only offered on host A and port 135/TCP is not offered on the network. Furthermore, it shows that the adversary was returned the application banner `Apache 2.0.54 (Debian GNU/Linux)` that reveals the web server type/version and operating system of the underlying platform. The use of an enhanced NEM allows us to determine what information about the network was obtained by the attacker and provides a measure to determine how effective the scanning activity was. Specifically, in this scenario four unique DCAs were sent into the network and only a single response was returned to the scanning host.

4.1.5 Bro Implementation

To implement the exposure maps technique, we used Bro [45] the open-source network -based IDS (see Section 3.1). Our Bro implementation of the exposure maps algorithm employs two custom policy scripts and modifies four generic policy scripts supplied by the Bro framework as follows (see Figure 4.8).

i. `darkport.bro`: a custom policy script that implements the exposure maps technique.

ii. `darkport-sig.sig`: a custom signature file that contains the regular expressions of known application string patterns (i.e. application banners) to detect transmitted application information. The signature file is extensible and new regular expressions can be added to identify additional applications.

iii. `brolite-dp.bro`: a modified version of the `brolite.bro` policy script that loads the required custom and modified Bro policies to implement the exposure maps technique.

iv. `signatures-dp.bro`: a modified version of the `signatures.bro` policy that interprets the `darkport-sig.sig` policy.

v. `trw-impl-dp.bro`: implements a modified `trw-impl.bro` policy augmented by using the exposure maps algorithm as a substitute method to determine connection success or failures.

vi. `local-dp.bro`: contains a modified version of the `local.bro` policy script that specifies the local network IP address information. It allows the user to precisely specify a range of IP addresses that will compose the local network subnet or subnets. A Bro-defined module `is_local_addr` makes use of the local network information to determine if the IP address belongs in the local network.

We make use of three generic Bro events (`connection_established`, `UDP_reply`, and `new_connection`) in our custom policy scripts to implement the exposure maps technique. More specifically, the NEM is constructed using information obtained by intercepting the Bro `connection_established` and `UDP_reply` events. The `connection_established` event indicates that a TCP three-way handshake has occurred and therefore a host has responded to a connection request on a specific TCP port. The `UDP_reply` event is used to determine the first occurrence of a UDP packet sent in response to an incoming UDP packet indicating a host has responded on a specific UDP port. In both cases, we extract the destination IP, port, and protocol information from the connection for consideration as a NEM entry. To determine when new incoming connection attempts are occurring that need to be checked against the NEM, we intercept the Bro `connection_attempt` event. Again, the destination IP, port, and protocol information is extracted from the connection that generated this event for comparison against the NEM. If these features match a NEM entry, it is considered a valid connection attempt; if not, it is considered a DCA.

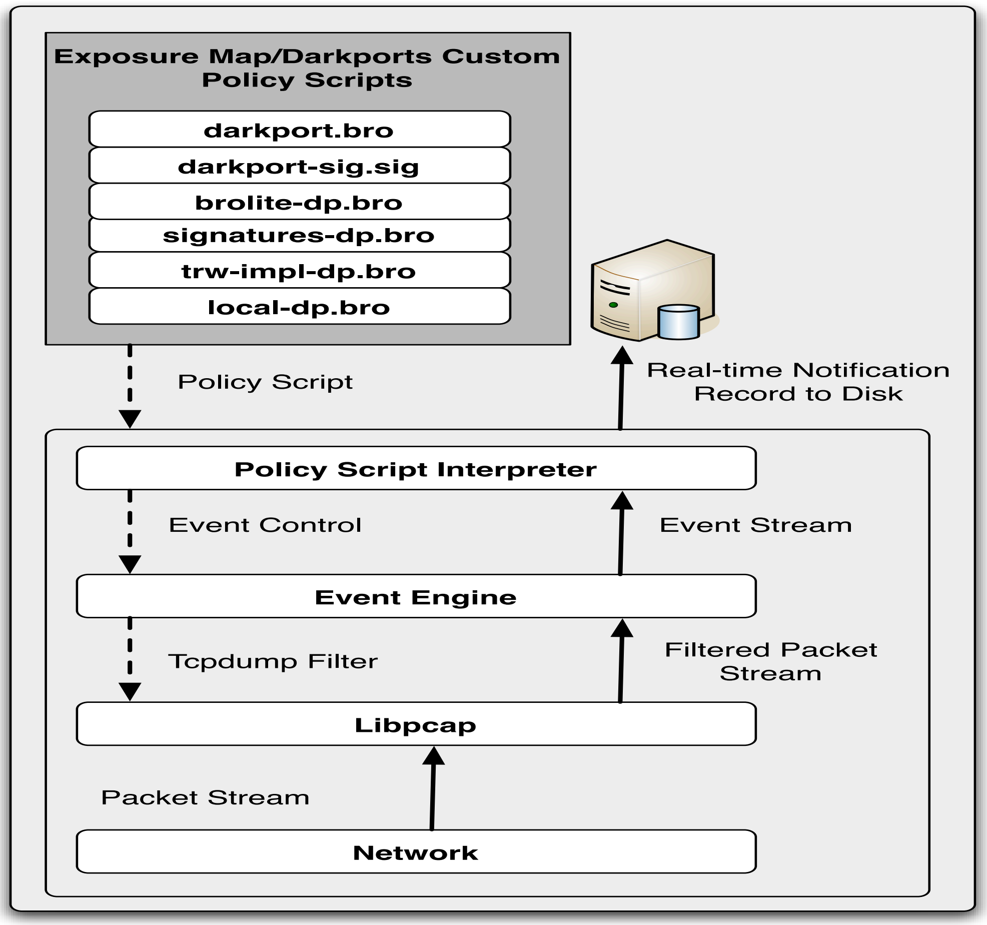


Figure 4.8: Bro Architecture Implementing Exposure Maps Policies.

To save information in state between different invocations in Bro (e.g. hardware replacements, software upgrades) the `persistent` attribute is used. Any information contained in a specific variable or data structure that is identified as being persistent is written to disk and loaded when the next invocation of Bro occurs. This feature is useful if for any reason monitoring is stopped for a period of time (e.g. software or hardware upgrades) and then restarted. In order to remove the values from specific variables or elements in a data structure, we make use of the `read_expire` and `create_expire` attributes. Specifically, the `read_expire` deletes a value or element from the respective data structure when the specified amount of time has elapsed since the element was accessed, while the `create_expire` performs a deletion after a specified amount of time has lapsed since the value or element was created.

Our `darkport.bro` policy script uses three main data structures:

- `services`: a persistent set that holds the NEM information. Entries in the NEM are automatically removed after a configurable amount of time using the `read_expire` attribute (the default is two weeks).

- `scanners_list`: the set of remote IP addresses that have been identified as performing scanning activity against the local network. Elements in this set are maintained for a configurable amount of time using the `create_expire` attribute (the default is 48 hours) and then removed.

- `services_appinfo`: the set that holds the enhanced NEM populated with application information obtained from the signatures contained in the `darkport -sig.sig` file. Entries in the enhanced NEM are automatically removed in accordance with the `read_expire` attribute set for the basic NEM.

The darkport policy script utilizes four distinct notifications and alerts that are respectively written to both the `notice.log` and `alert.log` files:

- NewService: a notification that a new service (i.e. trans-darkport) in the local network has been detected and is a candidate for inclusion in a vetted NEM. In the case of an unvetted NEM, inclusion is automatic.

- DCA: an alert indicating a DCA event has been detected. The alert contains the 5-tuple of information that characterizes the DCA.
- DCADeletion: a DCA alert has been generated as a result of trans-darkport activity from an unvetted NEM and is a candidate for deletion.
- PossibleTargetedScan: an alert that a source IP address in the `scanners_list` has made a connection attempt to an IP port number/protocol that matches an entry in NEM.

The training period for a vetted NEM is implemented by setting the `learning_mode` parameter to TRUE. This allows the NEM to be populated with IP addresses and associated port/protocol information but suppresses both inclusion of the source IP addresses of possible scanners into the `scanners_list` and all alert types. Once the training period has concluded, the `learnng_mode` parameter can be set to FALSE and the `scanners_list` and the alerting functions become active. The training period allows the necessary bootstrapping to occur to populate the NEM for vetting. In the case of an unvetted NEM, the `learning_mode` parameter is set to FALSE and not used.

Additionally, the `darkport.bro` policy script allows the user to specify what exposure maps/ darkports capabilities to use. Although the default is all (i.e. see Sections 4.1.1 – 4.1.4), the active response and RAA capabilities can be turned-off. This should significantly reduce system resource usage (see discussion in Section 5.2) as the need to construct and maintain both the scanners list and the enhanced NEM entries is not required.

Bro has the facility to write information to log files based on user-defined schedules. We generate three reports on a daily basis: (1) `nem.log`: the NEM entries, (2) `scanners_list.log`: the list of remote IP addresses detected undertaking possible scanning the local network, and (3) `possible_targeted_scanners.log`: the list of remote IP addresses that are suspected of previously performed reconnaissance activities against the network and are attempting to connect to an IP address/port combination in the NEM. Any associated application information obtained from the enhanced NEM (if used) is also recorded with each DCA.

The `trw-impl-dp.bro` policy makes use of the TRW algorithm (see Section 3.1) and augments it by including the NEM data structure exported from the `darkport.bro` policy. Herein, we refer to our implementation of the TRW algorithm augmented with a NEM as the *modified TRW* technique. Specifically, we use the NEM as a second connection oracle (NEM-oracle) to perform an additional check to determine if the connection will be a failure or success. If the destination IP and destination port/protocol of the incoming connection matches an entry in the NEM, the connection is assumed to be a success and the function returns a value of false (i.e. not a scan). If there is no match, the function proceeds and uses its own connection oracle (the original TRW-oracle) to determine if the connection is a success or failure. If the TRW-oracle also determines the connection is a failure then a possible scan against the network has been detected and the relevant information is updated. If the connection is considered a success and the destination IP is located in the local network, the NEM is updated accordingly. The modified TRW technique makes use of all the parameters (i.e. θ_0, θ_1, α, and β) associated with the TRW algorithm and updates its likelihood ratio, when a new connection attempt success or failure is observed, for comparison with thresholds η_1 and η_0. The internal logic of the modified TRW is the same as the original TRW algorithm with the exception of the addition of a NEM which is used as a secondary check to determine if a new connection attempt is a success or failure.

4.2 Evaluation: Datasets and Methodology

We evaluated the Bro implementation of our exposure maps technique using network traces obtained from two different network environments. The three datasets in total were taken from a small secure network (i.e. Carleton Computer Security Lab – CCSL) and a medium-sized enterprise network (i.e. the LBNL/ICSI 12-15 and LBNL/ICSI 12-16 datasets described below). Table 4.1 contains selected characteristics about these network traces.

CCSL Dataset. The CCSL network is a small university research network of 62 Internet reachable addresses connected to a university Internet accessible Class B network. All systems access the Internet through a firewall not permitting inbound connections unless initiated by an internal system. The CCSL dataset is a network trace archive that consists of months of full take (i.e. packets headers and content) capture files. To test our exposure maps technique, we used four weeks (September 2006) of network traces from this repository.

LBNL/ICSI Dataset. We arbitrarily selected two days worth of packet traces from the Lawrence Berkeley National Laboratory and International Computer Science Institute (LBNL/ICSI) Enterprise Tracing Project [46]. The packet traces were collected hourly and consist of anonymized packet headers with the content removed. The anonymization policy used to obfuscate the source and destination IP addresses as well as the tool used to perform the anonymization the tool (i.e. `tcpmkpub`) are described by Pang et al. [43].

This publicly available data set provides a corpus of network traffic for security researchers to use in the evaluation of new network-based detection algorithms. Unfortunately, the data capture and anonymization methodology has introduced both unwanted artifacts and gaps in the network traces [42]. Perhaps the three most relevant examples of these issues stem from the limitations of the system used to perform the network data captures, the need to preserve of anonymity of internal system addresses in the presence of certain forms of scanning activity, and the blocking of external scanning activity at the LBNL network border. We now describe these issues in turn.

The datasets in the LBNL/ICSI repository were obtained from two internal locations at the central routers. The collection system consisted of a commodity PC running the FreeBSD operating system and four network interface cards (NICs). Each NIC received unidirectional traffic streams that were merged using timestamp synchronization implemented in a custom NIC driver. The collection system could only monitor 2 of the over 20 routers ports at any given time, thus limiting the ability of the collected traffic to give a comprehensive view of the enterprise network activity. Additionally, network traffic was collected to and from the individual subnets but not inside the subnets thereby preventing any analysis of host activity within a given subnet. Finally, although no packet drops were reported from the collection system, subsequent analysis of TCP traffic revealed that some packet loss had occurred (see discussion in Section 5.1.1).

The second issue deals with the possibility that if a consistent mapping of IP addresses is used during anonymization, it may be possible to derive the original host and subnet addressing scheme by observing the order of internal hosts accessed during a sequential scan. In order to preserve the IP address anonymization undertaken in the network traces, scanners that sequentially probed a large number of IP addresses were identified and the associated network traffic was placed in network trace files separate from the non-scanning network traffic. The destination IP addresses probed by the scanners were then mapped with a separate subnet distinct from the destination IP addresses associated with non-scanning activity. This remapping results in a scanner's IP address being kept consistent within the trace but the target destination IP addresses of the scans are mapped using a different subnet than when the same destination IP address is associated with non-scanning activity. Remote hosts were classified as scanners if they attempted to access a minimum of 20 distinct systems in which at least 16 were in strictly descending or ascending order by IP address. An unfortunate consequence of mapping the destination IP addresses associated with the scanning activity using different subnets is that this makes it impossible to determine the total number of distinct internal IP addresses scanned by a particular scanner. However, this limitation of the datasets does not adversely affect the overall testing results of our evaluation. Specifically, the determination of the total number of distinct IP addresses scanned by a particular scanner was not part of the overall evaluation criteria. In our evaluation, we took all the individual normal and scanning one-hour packet traces for each day and merged them together into two trace files (i.e. LBNL/ICSI 12-15 and LBNL/ICSI 12-16) for analysis purposes.

The third and final issue is a result of the scan filtering performed at the LBNL gateway that blocks most TCP external scanning activity prior to data capture and subsequent inclusion in the datasets (see [42], section 3, page 3). As a result, a substantial amount of external TCP scanning activity directed at the network was never included in the archived datasets. Accordingly, the use of the LBNL/ICSI datasets in our evaluation introduces the real possibility that our results may unfairly favor the exposure maps technique and additional evaluations using more realistic

Table 4.1: Details about the Network Datasets.

	CCSL	LBNL/ICSI 12-15	LBNL/ICSI 12-16
Number of Packets	44 278 382	66 719 636	2 938 133
Trace Start	Sep 1, 00:46-06	Dec 15, 03:08-04	Dec 16, 11:15-04
Trace End	Oct 1, 00:47-06	Dec 16, 01:46-04	Dec 16, 22:19-04
Contiguous	Yes	Yes	Yes
Packet Payload	Yes	No	No
Local Network Subnets	10.0.0.0/192	128.3.0.0/16 131.243.0.0/16	128.3.0.0/16 131.243.0.0/16
Unique IPs (all)	13 930	25 434	20 948
Non-local Unique IPs	13 866	14 334	12 297

Table 4.2: Exposure Maps Capability Evaluation.

	CCSL	LBNL/ICSI 12-15	LBNL/ICSI 12-16
Scanning Detection	Yes	Yes	Yes
Advanced Scanning Detection	Yes	No	No
Active Response	Yes	No	No
Exposure Profiles	No	Yes	Yes
RAA	Yes	No	No

datasets may cause the exposure maps technique to generate more false positives.

Evaluation Methodology. To test the scalability of the various applications of exposure maps, it was important to understand how they would react in large network environments with a diverse user population using a variety of software applications. Accordingly, we evaluated the scalability of the prototype on the LBNL/ICSI datasets. The volume of network traffic and diversity of network activity in these datasets makes them ideal to test the scalability of the NEM with respect to scanning detection and the logical grouping of hosts into exposure profiles. As these datasets only contain anonymized packet headers and each merged packet trace spans approximately 24 hours, the concept of training period was not applicable. Therefore, during the scanning detection capability tests, an unvetted NEM was used. Furthermore, as there is no content within the payload of the anonymized packets, testing of the RAA capability was not possible using these datasets. Finally, the LBNL/ICSI datasets contains internal enterprise network traffic. To test the R2L scanning detection capabilities of our exposure maps technique, we selected two network subnets and defined these as the local network (see Table 4.1). Any IP addresses outside of these two subnet ranges (although still inside the network) are considered to be remote hosts. Accordingly, any scanning activity between local hosts (L2L) or from local hosts to remote hosts (L2R) will not be detected.

To complement these tests, we tested scanning (simple and advanced) detection, selected automated response, and RAA capabilities using the CCSL network dataset. The contiguous time period this dataset spanned (i.e. 28 days) allowed us to undertake a training period to make use of a vetted NEM and perform greater scan detection analysis to confirm our experimental results when evaluating the actual capabilities of the technique. As the network boundaries for this dataset are known, this allowed us to validate the NEM against a known network security policy. Additionally, having access to the full network traces allowed us to extract the necessary application layer information from the packet payloads to test the RAA capabilities of the technique. A summary of the specific exposure maps capabilities tested with each dataset is included in Table 4.2.

Chapter 5

Exposure Maps: Evaluation and Discussion

Our analysis herein of exposure maps is based on the experimental results observed through of a series of tests with three network datasets, and includes a side-by-side comparison with the TRW algorithm (see Section 3.1). This work was undertaken to show how exposure maps can be used to detect sophisticated scanning activity; analyze the effectiveness of using the exposure maps scanning detection capabilities to perform a real-time fine-grained automatic response to attacks; validate the network discovery and asset identification feature of exposure profiles/darkports in a mid-sized enterprise network environment; and verify the technique can perform an RAA as well as determine the overall network service information revealed by the network. We conclude with an examination of the limitations of the technique (Section 5.3), a brief summary of our approach and noteworthy experimental results (Section 5.4).

5.1 Evaluation of Exposure Maps

To validate the scanning detection capability of the exposure maps technique, we compared our results from both the CCSL network and LBNL/ICSI datasets with: (1) the TRW algorithm as implemented in the Bro policy `trw-impl.bro` as described by Jung [28], and (2) our modification of the TRW algorithm, which is augmented with an exposure maps connection oracle (i.e. NEM-oracle, as described in Section 4.1.5).

To set the TRW parameters for the CCSL network environment we considered both the number of active hosts (host density) and services (service density) in the network. The exposure maps technique provides both the number of active hosts on the network (which is the number of HEMs in the network, i.e. 3 in our case) and the services offered in the network as represented by the number of active IP/port combinations (which is the number of entries in the NEM that contain TCP ports, i.e. 7 in our case). We restricted the service density count to consider TCP ports exclusively for our comparison as the TRW implementation used for comparison only detects TCP scanning. To calculate a suitable value for θ_1 based on host density, we used the ratio of active hosts to total hosts in the network, .04838 (i.e. $\theta_1 = 3/62$). To calculate the second value of θ_1 based on the service density of the network we analyzed the data traces to determine the number of unique port/IP combinations probed by remote hosts. Specifically, we manually inspected the CCSL data traces using both the output of the Bro `conn.log` file and the exposure maps technique and determined that remote hosts made connection attempts to a total of 2 037 unique TCP port/IP combinations within the CCSL network over the four week period. Thus, the value of θ_1 based on service density is .00344 (i.e. $\theta_1 = 7/2\,037$). Our side-by-side comparison with TRW and the modified TRW technique consisted of two tests using the above mentioned host and service density values for θ_1 while θ_0 was set to .80 based on the analysis of Jung [28]. To set the TRW parameters for the LBNL/ICSI dataset, we selected the values determined by Jung [28]: $\theta_0 = .80$, $\theta_1 = .20$, $\alpha = .01$, and $\beta = .99$.

Evaluation Limitations. In order for the proper interpretation of our evaluation results to be made, two limitations of our side-by-side comparison of exposure maps, the modified TRW and the Bro TRW implementation are now discussed. The first limitation of our evaluation stems from the fact that the exposure maps technique does not take into consideration the source of the connection attempts when recording DCAs (see Section 4.1.1 for details). Therefore, unlike TRW, successful connection attempts are not individually tracked by remote hosts and thus are not a factor in the identification of scanning activity. Accordingly, there is no concept of a likelihood ratio in exposure maps nor do they attempt to identify benign hosts based on connection behavior. It is possible that non-malicious remote hosts may also possibly generate DCAs for a variety of reasons (e.g. mis-configuration, attempted connections to a relocated or no longer offered service). Depending on the heuristic used to process the DCAs, a legitimate remote host that makes a few unsuccessful connection attempts to the local network could be misclassified by the exposure maps technique as a scanner thereby generating a false positive (see discussion below).

Given the limitation of the exposure maps technique in its inability to track successful connections, the heuristic we developed to analyze DCAs for our comparison with TRW (herein referred to as the *Exposure Maps DCA Heuristic*) is confined to considering the number of failed connections a given remote host makes to unique local hosts per a predetermined time window in the absence of any successful connections. Furthermore, the Bro TRW implementation we used in our evaluation only considers failed connections to unique local hosts, repeated failed or successful connections to the same local host are simply ignored (see Section 3.1). Thus, we need to determine the minimum number of failed connection attempts (in the absence of successful connection attempts) to unique local hosts that need to occur for TRW to declare a remote host a scanner for a given set of parameters. Using the parameters selected for TRW on the CCSL dataset (i.e. the values of θ_1 that reflect both the host and service density, $\theta_0 = .80$, $\alpha = .01$, and $\beta = .99$) the expected number of observations (i.e. connection attempts), as defined by equation 3.8, that would be needed to confirm a remote host is a scanner by TRW is $E[N|H_1] = 3.339$ when θ_1 is set to the host density and $E[N|H_1] = 2.847$ when θ_1 is set to the service density. Using the logic in the `trw-impl.bro` Bro policy of Section 3.1, the threshold η_1 will be reached, implying that a remote host is classified as a scanner, if a remote host makes failed connection attempts to at least 3 unique local hosts in the absence of any successful connection attempts (see Figure 3.1; note $\Lambda(r) = (\frac{1-\theta_1}{1-\theta_0})^x \geq .0101$ in equation 3.3 for $x \geq 3$). This is in terms of the values of θ_1 based both on the service and host density of the network.

Using the parameters selected for TRW on the LBNL/ICSI dataset (i.e. $\theta_1 = .20$, $\theta_0 = .80$, $\alpha = .01$, and $\beta = .99$) the expected number of observations (i.e. connection attempts), as defined by equation 3.8, that would be need to be observed to confirm a remote host is scanner is $E[N|H_1] = 5.414$. Threshold η_1 will be reached such that a remote host is determined to be a scanner if it makes failed connection attempts to 4 unique local hosts (i.e. using the logic in the `trw-impl.bro` Bro policy resulted in $(\frac{1-\theta_1}{1-\theta_0})^4 \geq \eta_1$; see equation (3.3)) in the absence of any successful connection attempts. Herein, we define the minimum number of unique local hosts a remote host must make failed connections to in order to be considered a scanner by the exposure maps

Table 5.1: Parameters Used for Comparing TRW to the Exposure Maps DCA Heuristic (CCSL Dataset).

	uniq-localhosts-scanned	θ_0	θ_1	β	α
Exposure Maps	3	n/a	n/a	n/a	n/a
TRW θ_1 Host Density	n/a	.80	.04838	.99	.01
TRW θ_1 Service Density	n/a	.80	.00344	.99	.01

Table 5.2: Affect of Hits and Misses on TRW Hypotheses Selection. (θ_0 = .80, θ_1 = .20, α = .01, β = .99).

Remote Host (RH)	**Connection Attempts**							Selected Hypothesis
RH1	H	H	H	**H**				H_0 – Benign
RH2	H	M	H	H	H	**H**		H_0 – Benign
RH3	M	M	M	**M**				H_1 – Scanner
RH4	H	M	H	M	M	H	M	Undetermined
RH5	M	M	M	H	M	**M**		H_1 – Scanner

DCA heuristic we developed for our comparison with TRW as *uniq-localhosts-scanned*. Table 5.1 shows the parameters used by the exposure maps technique, the modified TRW technique, and TRW for our evaluation with the CCSL dataset. Both the TRW and modified TRW techniques used the TRW parameters associated with the service and host density values for θ_1 listed in the table.

A remote host will be classified as benign if it makes a number of successful connection attempts to unique local hosts (using the Bro TRW implementation) such that the likelihood ratio drops below the threshold η_0. Specifically, when $\theta_1 = .04838$, the expected number of observations (i.e. connection attempts) to determine that a remote host is benign is $E[N|H_0] = 2.330$ and this value drops to $E[N|H_0] = 1.115$ when when $\theta_1 = .00344$ (see discussion in Section 5.3). A remote host is considered *undetermined* (i.e. could be a scanner or benign) if it neither crosses above threshold η_1 nor below η_0 within the predetermined time window. The desired detection rate and false positive probability parameters in the policy were set to .99 and .01 respectively. Again, to set the TRW parameters for the LBNL/ICSI dataset, we selected the values determined by Jung [28]: $\theta_0 = .80$, $\theta_1 = .20$, $\alpha = .01$, and $\beta = .99$. Thus, for the LBNL/ICSI datasets, the number of expected observations required to classify a remote host as benign is $E[N|H_0] = 5.414$.

Table 5.3: Expected Number of Connection Events ($E[N|H_1]$) to Confirm a Remote Host is a Scanner With $\alpha = .01$ and $\beta = .99$ Kept Constant.

	$\theta_0 = 0.6$	$\theta_0 = 0.7$	$\theta_0 = 0.8$	$\theta_0 = 0.9$
$\theta_1 = 0.01$	5.259327	3.951920	2.924977	2.024272
$\theta_1 = 0.05$	6.456205	4.675788	3.356582	2.258148
$\theta_1 = 0.10$	8.177836	5.670416	3.930451	2.561878
$\theta_1 = 0.15$	10.405763	6.883874	4.600825	2.904754
$\theta_1 = 0.20$	13.450660	8.431242	5.413975	3.304537
$\theta_1 = 0.25$	17.828219	10.477150	6.428308	3.781217
$\theta_1 = 0.30$	24.502386	13.286996	7.728387	4.361244
$\theta_1 = 0.35$	35.477595	17.321938	9.444900	5.082008
$\theta_1 = 0.40$	55.531504	23.449129	11.791352	5.998822
$\theta_1 = 0.45$	98.554592	33.468740	15.138563	7.197142
$\theta_1 = 0.50$	220.627018	51.656208	20.180809	8.815567

TRW provides the capability of declaring a remote host a scanner after some combination of successful and unsuccessful connection attempts. The exposure maps DCA heuristic we developed to compare exposure maps with TRW does not take into consideration any successful connection attempts and thus any intermingled successful connection attempts by a remote host are simply ignored. Table 5.2 provides illustrative examples of how successful and failed connection attempts affect the selection of the hypothesis that a remote host is performing scanning activity. An H or hit represents a successful connection attempt, while an M or miss represents an unsuccessful connection attempt. A shaded table cell that contains an H or M represents when a hypothesis (i.e. H_0 (benign host) or H_1 (scanner)) was selected by TRW based on the connection history of the remote host to the local network. In one case, there is not enough evidence to declare a remote host a scanner or benign and in this case the hypothesis about the remote host is classified as undetermined (e.g. RH4 in Table 5.2).

Due to the inability of exposure maps to track successful connection attempts, there may be some instances where a remote host would be classified as either benign or undetermined by TRW and a scanner by the exposure maps technique. For instance, using the exposure maps DCA heuristic we developed (i.e. *uniq-localhosts -scanned* ≥ 4 within the predetermined time window for the LBNL/ICSI dataset) to

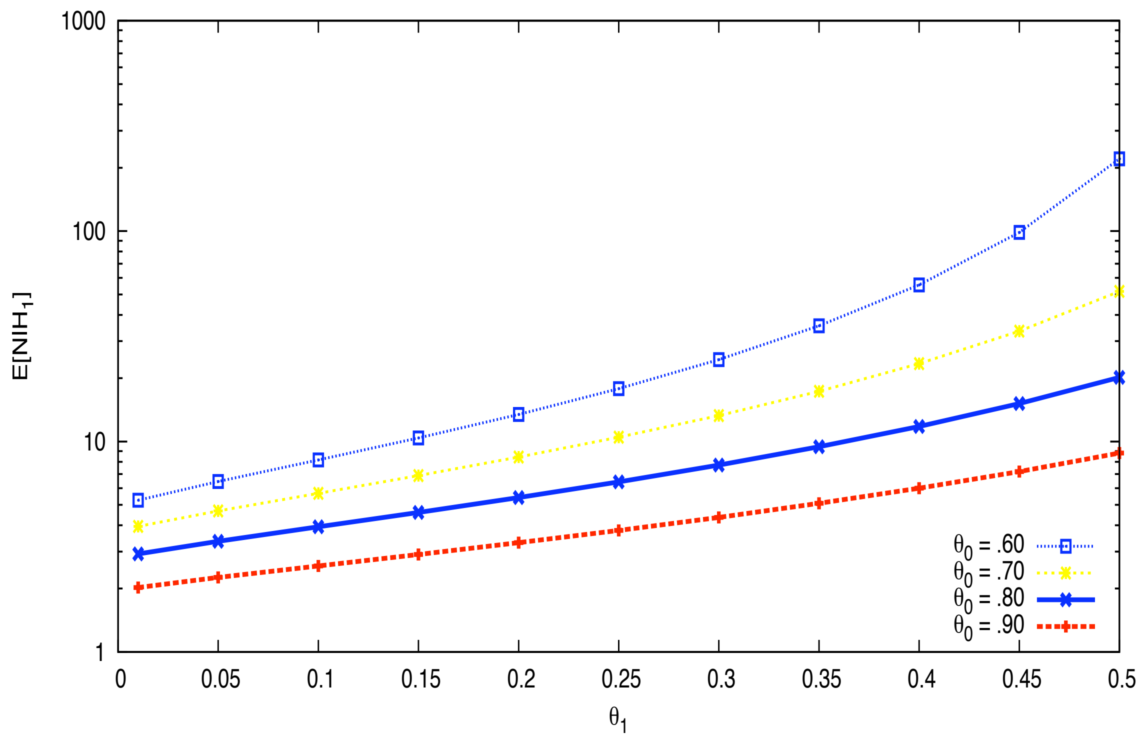


Figure 5.1: Expected Number of Connection Events to Confirm a Remote Host is a Scanner With $\alpha = .01$ and $\beta = .99$ Kept Constant.

compare the scanning detection capability of exposure maps with the Bro implementation of TRW, RH4 (see Table 5.2) would be classified as undetermined by the latter and a scanner by the former. The modified TRW technique used in our evaluation uses the NEM only as a second connection oracle (NEM-oracle) to perform an additional check to determine if the connection will be a failure or success (see Section 4.1.5). As it makes use of the TRW algorithm tracking both connection successes and failures, it does share this limitation with the exposure maps technique.

The second limitation of our evaluation stems from selection of the θ_0 value during our evaluation. A value of $\theta_0 = .80$ was selected for all our testing on all three datasets. The expected number of observations (i.e. connection attempts) required to confirm a remote host is either benign or a scanner can be determined by equation 3.8. Specifically, $E[N|H_1]$ is a function of four parameters, α, β, θ_0 and θ_1 which represents that false positive and detection probabilities and the degree to which benign hosts and scanners differ in their probability to generate failed connection attempts. The greater the number of accessible servers and services in a network, the greater the probability that a scanner will be able to successfully make a connection to a target. Accordingly, as the value of θ_1 increases, the greater the number of observations are required to declare a remote host as a scanner. Table 5.3 and Figure 5.1 show the affect of θ_0 and θ_1 (with $\alpha = .01$ and $\beta = .99$ held constant) on the expected number of observations required to classify a host as a scanner. A more rigorous analysis would explore a full range of values of both θ_0 and θ_1 for these datasets to determine the overall affect on the comparison of the three techniques.

Limited Ground Truth. In order to determine the false positives and negatives produced during the evaluation, it was necessary to gain a *limited ground truth* of the overall scanning activity present within the dataset. We define limited ground truth as confirmation of scanning results through the use of ancillary information. For the purpose of our evaluation, the ancillary information we used to determine the limited ground truth consisted of the output of the unmodified TRW implementation and manual inspection of the datasets used in the evaluation. Specifically, any unique scanning activity detected by any of the techniques (after careful analysis) would either result in a source of false positives (for the technique that detected the scan) if the activity was determined to be benign or a source of false negatives (for the technique that did not detect the scan) if the activity was determined to be part of a scanning campaign.

To determine the number of false positives generated during the evaluation, careful manual analysis of every scan detected by each technique was undertaken to confirm it was actual scanning activity. This manual analysis consisted of using the destination IP, source IP, and timestamp information from each scan to identify the associated network traffic (i.e. failed and successful connection attempts) for subsequent viewing and analysis using both `tcpdump`[2] and the relevant Bro `conn.log` files. Through user error or misconfiguration, a connection attempt might be made to a host or service not offered by the network. In this instance, the intent of the connection attempt was not to scan some portion of the network, but rather it is simply a failed attempt to access a legitimate service. Regardless, this activity would be classified as a DCA (or in the case of TRW a failed connection attempt) because an attempt was made to connect to a host/port pair not listed in the NEM. Again, given that there is no way to measure the *intent* of a connection attempt, the exposure maps technique will classify these events as DCAs.

Exposure maps (once vetted against the security policy) define the authorized access to the network from external sources. Connections attempts or scans outside these maps are considered a possible scan. Connection attempts that are part of a scan directed against a port/IP combination contained in the NEM would not be considered a DCA but rather a connection attempt to a valid service; this might potentially then be a source of false negatives, and to claim otherwise (i.e. zero false negatives in general) would imply unknowable knowledge of the intent of the party requesting the connection. For instance, a scan to port 443/TCP of host 10.0.0.2 (see Table 5.4) in the CCSL network would not be recorded as a DCA. In practice, although this specific event in a scanning campaign would not be detected, the overall scanning campaign would likely be detected using exposure maps for scan detection, as in most cases we would expect with high probability scans to occur against other hosts in the network not offering SSL (i.e. port 443/TCP darkports). This relies on the assumption that an attacker is not using a botnet to scan the network (see Section 2.2.2) thus making detection of the specific scanning campaign impractical. In this case, a network operator would have to rely on observing that an overall increase in DCAs to a specific port was occurring in order to provide an indication of changes in detected scanning behavior.

5.1.1 Results: Scanning Detection – Comparison With Threshold Random Walk (TRW)

To compare our exposure maps technique with the TRW algorithm as implemented in the `trw-impl.bro` policy, it was necessary to take into consideration two constraints: (1) the TRW algorithm only detects TCP protocol scanning activity, and (2) the TRW algorithm uses an observation time period (i.e. defaulted to 30 mins) that the exposure maps DCA heuristic must use when determining *uniq-localhosts-scanned*.

The `darkport.bro` policy implementation of our exposure maps technique detects both TCP and UDP scanning activity. We restricted our comparison with TRW to

Table 5.4: NEM Details for CCSL Network.

Host	TCP Ports	Description
10.0.0.1	25/TCP, 53/UDP, 631/TCP, 993/TCP	SMTP/DNS/IPP/IMAP
10.0.0.2	22/TCP, 53/UDP, 80/TCP, 443/TCP	SSH/DNS/HTTP/SSL
10.0.0.3	22/TCP	SSH

TCP scanning. An off-line parsing script to process the TCP DCAs generated by our technique that used (1) the same observation time period as the `trw-impl.bro` policy (i.e. 30 mins), and (2) using the thresholds for the exposure maps DCA heuristic (see Section 5.1, page 62) created for our evaluation derived from the TRW parameters described in Section 5.1 that require 3 *uniq-localhosts-scanned* per observation time window for the CCSL dataset and 4 *uniq-localhosts-scanned* per observation time window for the LBNL/ICSI datasets.

CCSL Network Dataset. The NEM for the CCSL dataset is comprised of three HEMs (see Table 5.4). Two of these have four active ports; the third has one active port. The NEM thus has in total nine port/IP entries. We used a one-day training period to construct the NEM; it stabilized within the first 20 hours of network traffic and vetting confirmed that no services or hosts beyond those allowed by the existing network security policy were detected.

Tables 5.5 and 5.6 summarize the results of the side-by side comparison of the exposure maps technique against the TRW and modified TRW (using a NEM-oracle) technique, for the CCSL dataset. The same results were obtained for exposure maps, Bro TRW, and modified TRW implementation and thus are only shown once in the table. The variable θ_1 used for the TRW and modified TRW techniques was set to reflect the host density ($\theta_1 = .04838$) and the service density ($\theta_1 = .00344$) of the network. The variable θ_0 was set to .80 for both tests. All three techniques declared 2 233 port scans. 279 remote hosts were determined to be scanners and 183 remote hosts could be classified as either undetermined or benign based on the TRW and modified TRW implementation. The exposure maps DCA heuristic detected no ambiguous connection attempts (see discussion of false negatives for the LBNL/ICSI datasets within this section).

Figure 5.2 shows the scanning activity in terms of the number of TCP and UDP

Table 5.5: Scanning Detection Results for CCSL Dataset with the Exposure Maps DCA Heuristic. A value of 3 *uniq-localhosts-scanned* per observation time window was used for the exposure maps DCA heuristic.

Exposure Maps DCA Heuristic	
Declared Scans	2 233
False Positives	0
False Negatives	0
Ambiguous Connection Attempts	0
Unique Scanners	279
Undetermined and Benign Hosts	n/a

Table 5.6: Scanning Detection Results for CCSL Dataset with the Bro TRW and Modified TRW Techniques. θ_0 was set to .80 for both tests.

Technique		Host Density $\theta_1 = .04838$	Service Density $\theta_1 = .00344$
Bro TRW	TRW-Scans Detected	2 233	2 233
	False Positives	1	0
	False Negatives	0	0
	Ambiguous Connection Attempts	n/a	n/a
	Unique Scanners	279	279
	Undetermined and Benign Hosts	183	183
Modified TRW	TRW-Scans Detected	2 233	2 233
	False Positives	1	0
	False Negatives	0	0
	Ambiguous Connection Attempts	n/a	n/a
	Unique Scanners	279	279
	Undetermined and Benign Hosts	183	183

DCAs generated by unique remote hosts as detected by the exposure maps algorithm. The y-axis is set to log scale. The number of unique remote hosts is the sum of unique IP addresses extracted from both the UDP and TCP DCAs for each day. TCP DCA activity dominates UDP DCA activity for the majority of the four week period.

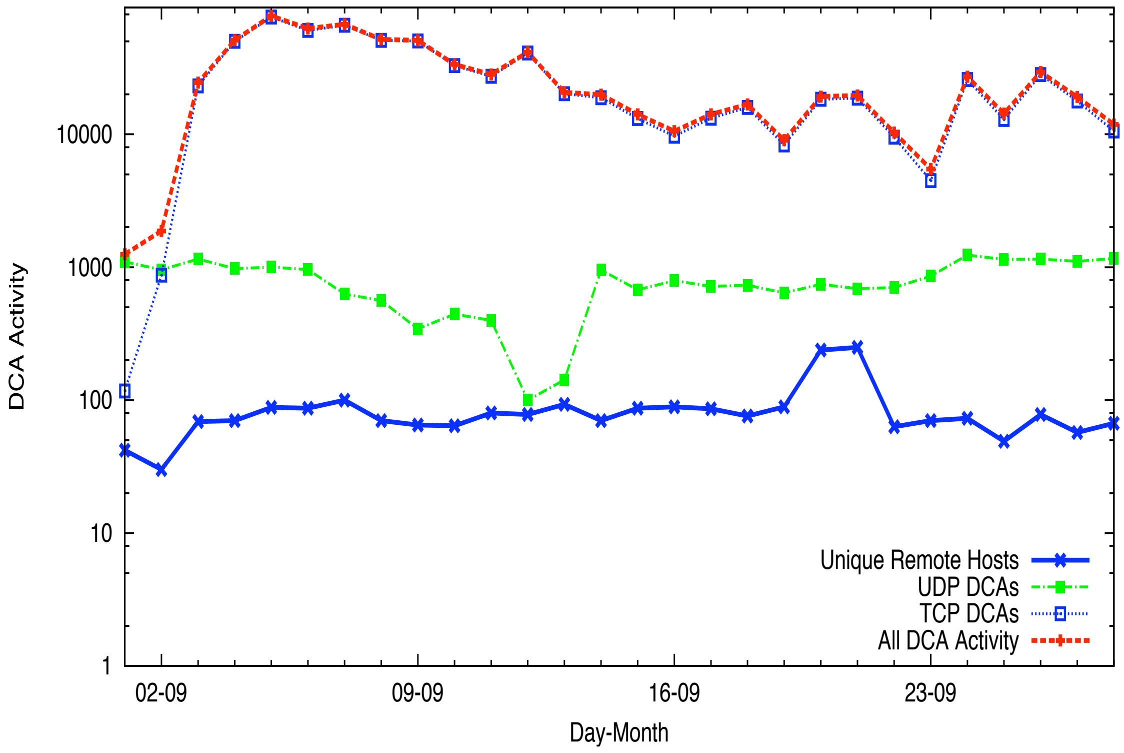


Figure 5.2: DCA Activity for CCSL Dataset (Using Exposure Maps Technique).

We extracted the source IP addresses of the detected DCAs and geolocated them using the MaxMind geolocation database [33] to determine the countries of origin. At the country level, a purported geolocation accuracy of 99% is claimed. Figure 5.3 shows that both Canada and the United States have the greatest number of observed remote hosts responsible for generating DCAs. The circles in the figure were arbitrarily placed on the capital city of the country of DCA origin. The Canadian IP addresses may be partially explained by the fact that the CCSL network is a small subnet of the much larger Class B Carleton University network. A number of the remote hosts have IP addresses that fall within this address range. This is not an unusual occurrence as a number of scanning techniques (typically worms and autorouters, see Section 2.2.1) have a bias to scan local network ranges as determined by the the scanning system's IP address. This affinity skews the geographic distribution of the remote hosts.

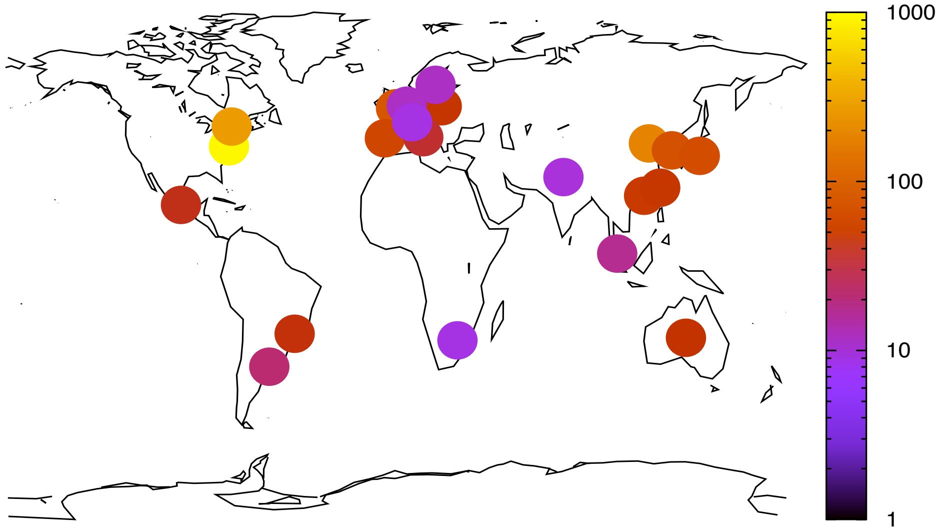


Figure 5.3: Number of Unique Remote Hosts Generating DCAs in the CCSL Dataset (Using Exposure Maps Technique) per Geo-located Country of Origin. (Figure requires color image.)

Figure 5.4 depicts the cumulative distribution function of the number of unique local hosts for which DCAs were generated by remote hosts detected with the exposure maps technique. Approximately 60% of the remote hosts generating TCP DCAs and 85% of the remote hosts generating UDP DCAs did so against a single local host. Approximately 35% of the remote hosts that generated TCP DCAs and less than 10% of the remote hosts generating UDP DCAs did so against three or more unique local hosts (i.e. the threshold determined for the exposure maps DCA heuristic used by the exposure maps technique). Two key observations from the DCA activity of remote hosts are: (1) remote hosts that generate TCP DCAs are more likely to do so against greater numbers of local hosts, and (2) a large number of remote hosts generated DCAs (both UDP and TCP) against a single local host.

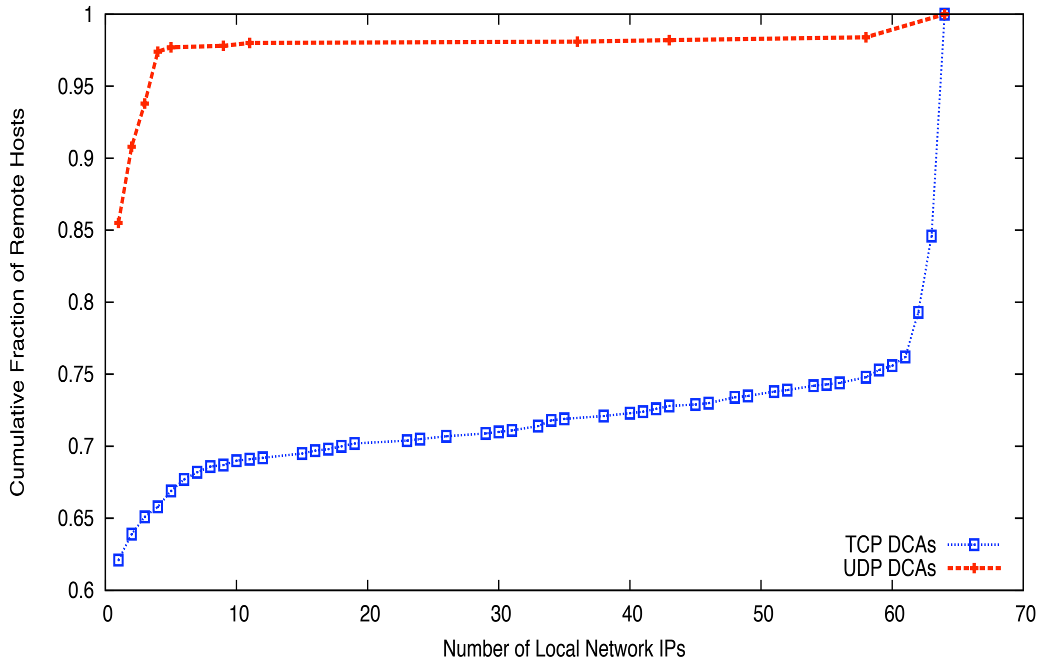


Figure 5.4: Fraction of Remote Hosts Generating DCAs Against at Most x Local Hosts. CCSL Dataset, Using Exposure Maps Technique.

Figure 5.5 depicts the cumulative distribution function (x-axis was set to log scale) of number of the DCAs generated by remote hosts detected with the exposure maps technique. It is interesting to note that less than 10% of the remote hosts generating TCP DCAs, and 80% of the remote hosts generating UDP DCAs, generated exactly one DCA. This would indicate that the approximately 50% of the remote hosts generating TCP DCAs against a single local host (recall that 60% of the remote hosts generating TCP DCAs did so against a single host) did so using more than one DCA. Relatively few DCAs detected from a remote host within a long time period may mean either a very slow scan or simply a false positive as a result of a system misconfiguration or backscatter (see false positive discussion below). The slope of the TCP DCA distribution is steeper than the UDP DCA distribution slope indicating that there is more variation for remote hosts generating UDP DCAs in the number of actual DCAs they generate. TCP DCA activity was the most prevalent observed in the dataset in terms of both number of unique remote hosts generating DCAs and

Table 5.7: Additional Exposure Maps Results for the CCSL Dataset.

Exposure Maps	
# of NEM Entries (TCP)	7
Total DCAs (TCP and UDP)	747 320
Unique Remote Hosts (TCP and UDP)	1 489

the overall number of DCAs. In fact, we detected three remote hosts that generated over 50 000 TCP DCAs each. Table 5.7 shows that 747 320 DCAs were associated with 1 489 remote hosts over the four week period.

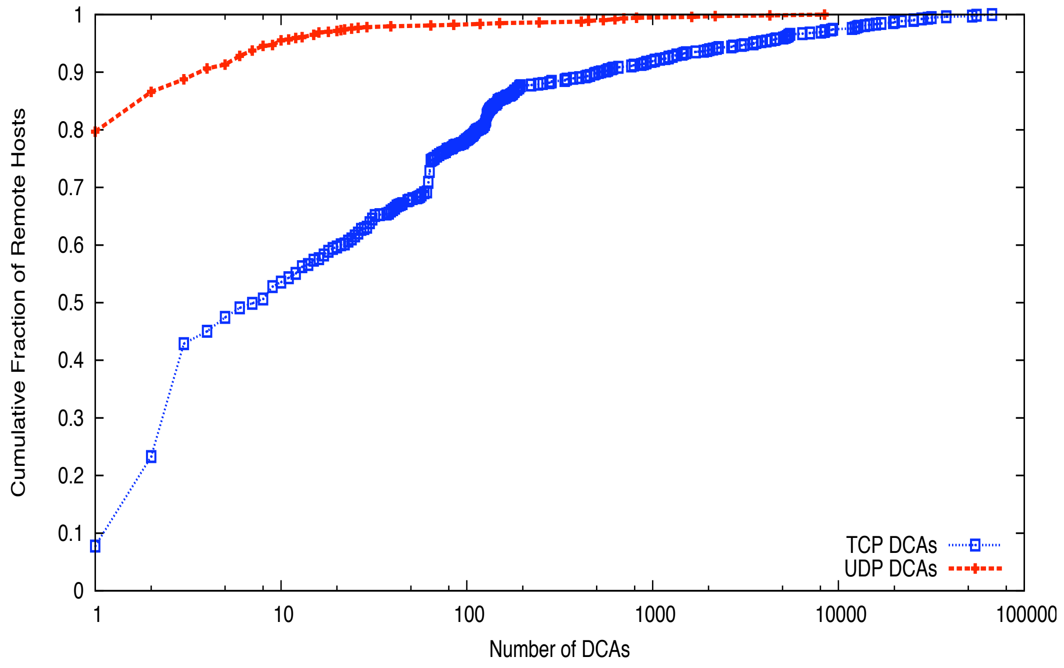


Figure 5.5: CDF of DCA Activity for the CCSL Dataset (Using the Exposure Maps Technique) – Number of DCAs.

False Negatives for the CCSL Dataset. We relied on the output of the unmodified TRW implementation and manual inspection of both the failed and successful connection attempts based on the output of the Bro `conn.log` file to provide a limited ground truth of the scanning activity present within the dataset (see Section 5.1). The exposure maps DCA heuristic detected all the scans identified by the TRW algorithm. Thus relative to TRW, for this dataset, our analysis for exposure maps

(Tables 5.5 and 5.6) revealed no false negatives. In terms of the false negatives analysis for the TRW and modified TRW techniques, this was determined by comparing, after careful analysis, the output of any unique scans detected by the exposure maps technique. Again relative to TRW, for this dataset our analysis revealed no false negatives. The modified TRW technique also generated no false positives or negatives for the dataset after careful analysis as previously described.

False Positives for the CCSL Dataset. The exposure maps DCA heuristic created for comparison with the TRW and modified TRW techniques would declare a remote host as a scanner when *uniq-localhosts-scanned* ≥ 3 within a finite time window. Remote host activity resulting in only 1 or 2 DCAs are below the minimum detection threshold for all three of the detection techniques used in our comparative evaluation (i.e. the minimum required observations to determine if a remote host is a scanner did not occur within the finite time window) eliminating these as a possible source of false positives during our evaluation. We recorded no false positives for the exposure maps algorithm, or for the TRW and modified TRW techniques over the two test runs. While no false positives occurred in our CCSL dataset test for the exposure maps technique, we do not claim this in general. Whenever a vetted NEM is used, false positives will be generated whenever new legitimate services are introduced on the network or services are utilized which were not accessed during the training period (with identification as a trans-darkport until the service has been added to the vetted NEM). We expect trans-darkports to occur infrequently in tightly controlled enterprise environments (e.g. in most government departments, financial, and health care). However, the use of an unvetted NEM provides a mechanism (i.e. an automated scan event deletion message – the DCADeletion notification (see Section 4.1.5)) to remove DCAs caused by trans-darkport activity.

LBNL/ICSI 12-15 Dataset. Figures 5.6 to 5.8 inclusive were generated using the results of the exposure maps technique on the LBNL/ICSI 12-15 dataset. Figure 5.6 shows the network DCA activity detected by the exposure maps technique for the LBNL/ICSI 12-15 dataset. The y-axis plots the number of DCAs set to log scale and the x-axis is set at hourly intervals. The amount of TCP DCA activity is dominant over UDP DCA activity for the majority of the dataset.

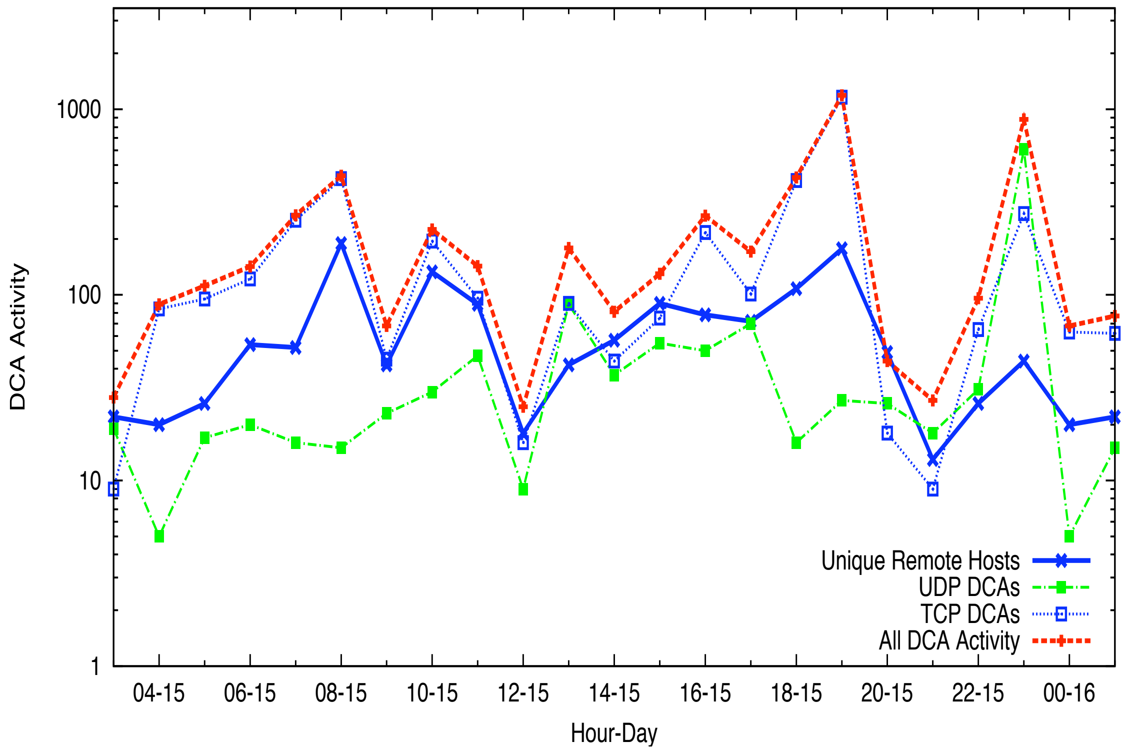


Figure 5.6: DCA Activity for LBNL/ICSI 12-15 Dataset (Using Exposure Maps Technique).

The cumulative distribution function of the number of unique local hosts that were the subject of DCAs by remote hosts detected with the exposure maps technique for the LBNL/ICSI 12-15 dataset is given in Figure 5.7. Approximately 82% of the remote hosts that generated TCP DCAs and 65% of the remote generating UDP DCAs did so only against a single host. Even the most active remote hosts responsible for both TCP and UDP DCAs generated DCAs against less than 80 unique local hosts in total out of approximately 11 000 possible local hosts.

Figure 5.8 depicts the cumulative distribution function of the number of DCAs generated by remote hosts detected with the exposure maps technique for the LBNL/ ICSI 12-15 dataset. Approximately 90% of the remote hosts generated less than 50 UDP and TCP DCAs each. The remote host with the most DCA activity generated a total of 1 161 TCP DCAs (with multiple DCAs detected against different ports on the same local hosts).

The NEM for the LBNL/ICSI 12-15 dataset consisted of 3 604 entries; 2 372 of

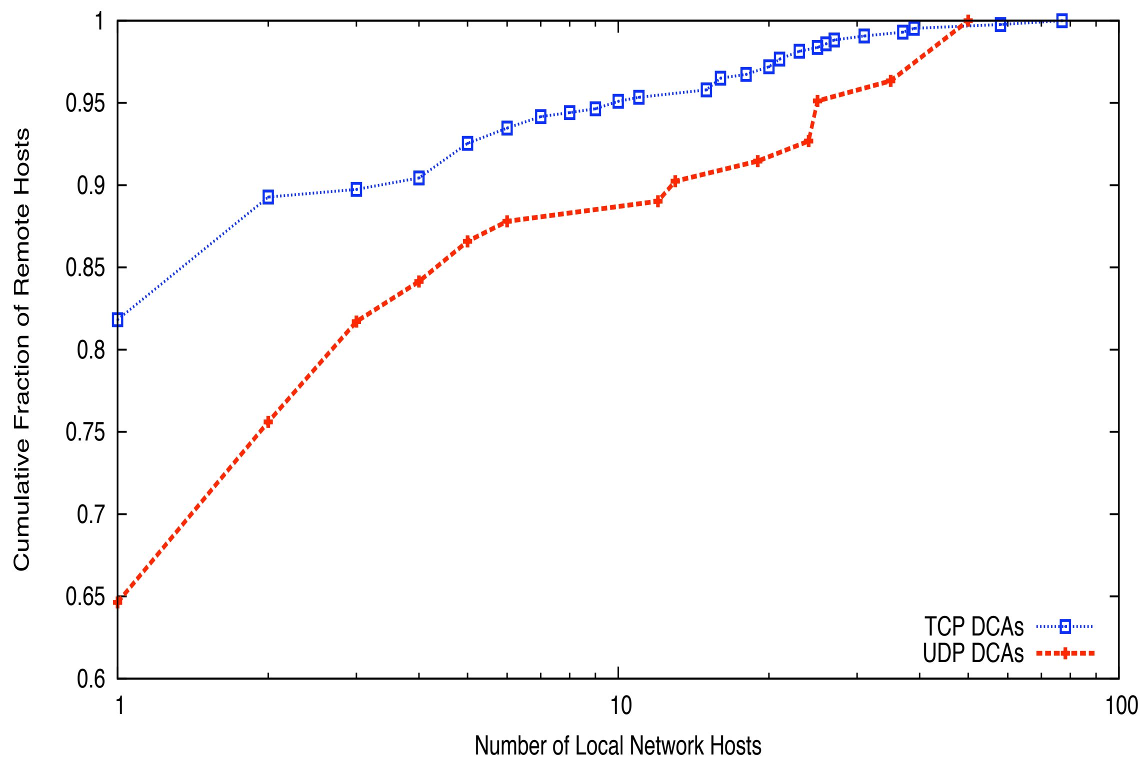


Figure 5.7: Fraction of Remote Hosts Generating DCAs Against at Most x Local Hosts. LBNL 12-15 Dataset, Using Exposure Maps Technique.

these were TCP ports (see Table 5.9). We used an unvetted NEM for our evaluation and therefore did not require a training period. Table 5.8 lists the results of the side-by side comparison of the exposure maps technique against the TRW and modified TRW (using a NEM-oracle) technique.

The exposure maps DCA heuristic detected 42 scans one of which we determined was an ambiguous connection attempt (see discussion of false negatives for the LBNL/ICSI datasets later within this section). The TRW and modified TRW technique both detected a total of 41 scans. The TRW and modified TRW technique both classified 67 remote hosts as benign. We can see that all three algorithms performed very well in terms of false positive and negative rates. The exposure maps technique generated no false positives or negatives. The TRW and modified TRW techniques each had a total of 1 false positive and no false negatives respectively. A discussion of how the false positive and negative rates for all three techniques were determined for both the LBNL/ICSI 12-15 and LBNL/ICSI 12-16 datasets is included

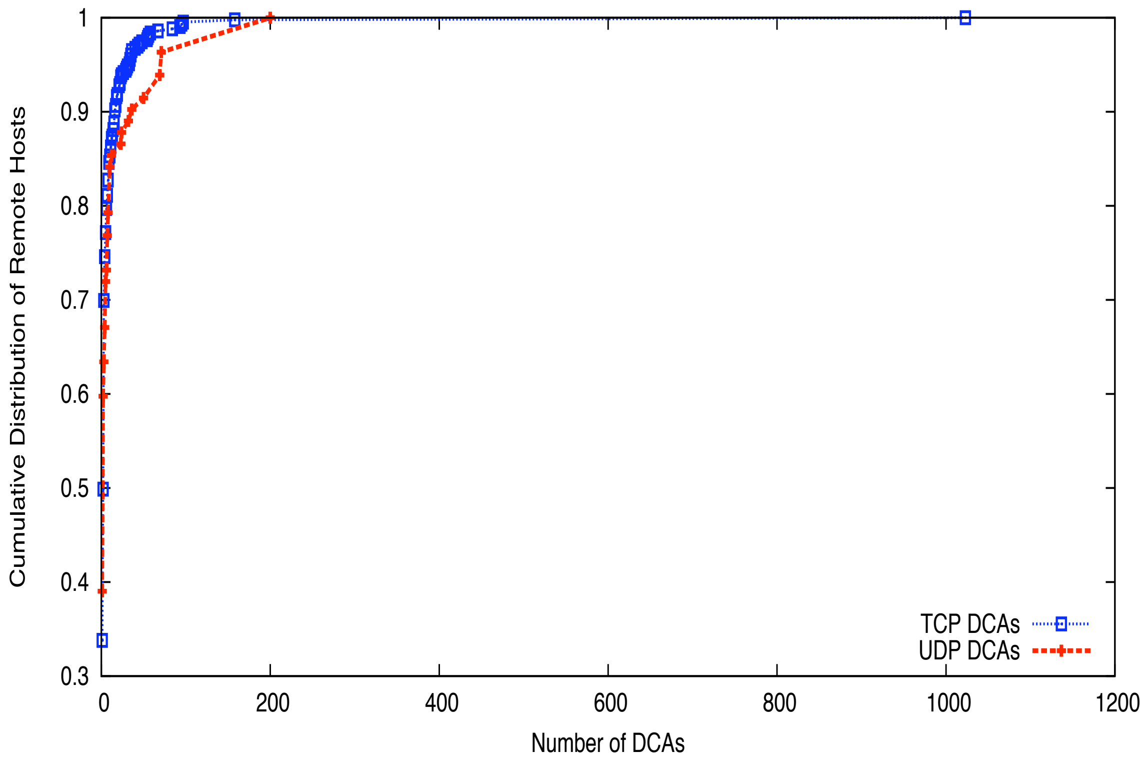


Figure 5.8: CDF of DCA Activity for LBNL/ICSI 12-15 Dataset (Using Exposure Maps Technique) – Number of DCAs.

at the end of this section. Table 5.9 shows 5 176 DCAs were associated with 501 unique remote hosts.

LBNL/ICSI 12-16 Dataset. Figures 5.9 to 5.11 inclusive summarize the results of the exposure maps technique on the LBNL/ICSI 12-16 dataset. Figure 5.9 shows the DCA activity detected by the exposure maps technique for the LBNL/ICSI 12-16 dataset. The y-axis plots the number of DCAs set to log scale and the x-axis is set at hourly intervals. TCP DCA activity volume dominates UDP DCA activity volume for the majority of the dataset.

The cumulative distribution function of the number of unique local hosts that were the subject of DCAs by remote hosts detected with the exposure maps technique for the LBNL/ICSI 12-16 dataset is given in Figure 5.10. Approximately 38% of the remote hosts generating UDP DCAs and 35% of the remote hosts generating TCP DCAs did so against a single local host. The most active remote hosts responsible for generating TCP and UDP DCAs respectively, did so against more than 1 000 and

Table 5.8: Scanning Detection Comparison LBNL/ICSI 2004-12-15 Dataset. The values $\theta_1 = .20$ and $\theta_0 = .80$ were used for the Bro TRW and modified TRW techniques and a value of 4 *uniq-localhosts-scanned* per observation time window was used for the exposure maps DCA heuristic.

Technique		
Exposure Maps	Declared Scans	42
	False Positives	0
	False Negatives	0
	Ambiguous Connection Attempts	1
	Benign Hosts	n/a
Bro TRW	TRW Scans	41
	False Positives	1
	False Negatives	0
	Ambiguous Connection Attempts	n/a
	Benign Hosts	67
Modified TRW	TRW Scans	41
	False Positives	1
	False Negatives	0
	Ambiguous Connection Attempts	n/a
	Benign Hosts	67

Table 5.9: Additional Exposure Maps Results for the LBNL/ICSI 2004-12-15 Dataset.

Exposure Maps	
# of NEM Entries (TCP)	3 604 (2 372)
DCAs - TCP	3 929
DCAs - UDP	1 247
Total DCAs (TCP&UDP)	5 176
Unique Remote Hosts (TCP&UDP)	501

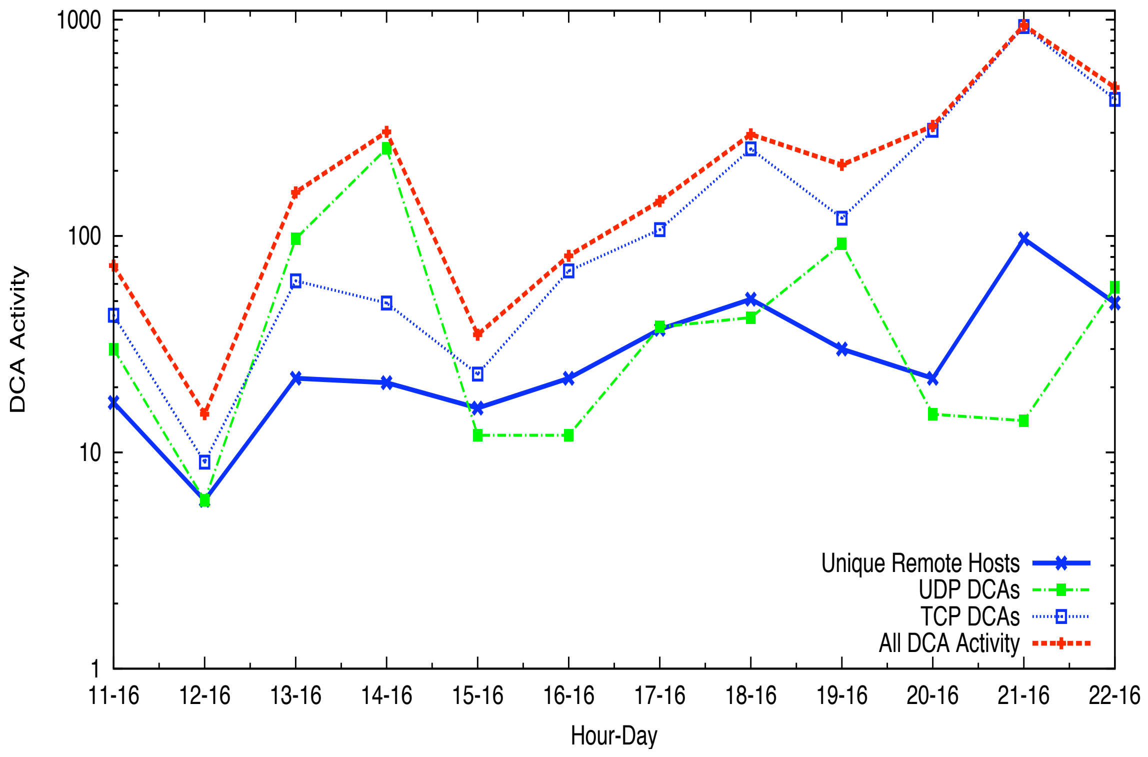


Figure 5.9: DCA Activity for LBNL/ICSI 12-16 Dataset (Using Exposure Maps Technique).

200 unique local hosts in total out of approximately 8 500 local hosts in the network.

Figure 5.11 depicts the cumulative distribution function of the number of DCAs generated by remote hosts detected with the exposure maps technique for the LBNL/ ICSI 12-16 dataset. Approximately 85% of the remote hosts that generated UDP DCAs and 90% of the remote hosts that generated TCP DCAs produced less than ten DCAs. The remote host with the most activity generated a total of 929 TCP DCAs.

The NEM for the LBNL/ICSI 12-16 dataset consisted of 2 695 entries; 1 652 were TCP ports (see Table 5.11). We used an unvetted NEM for our evaluation and therefore did not require any training period. Table 5.10 lists the results of the side-by side comparison of the exposure maps technique against the TRW and modified TRW (using a NEM-oracle) technique. The exposure maps DCA heuristic detected 17 scans one of which we determined was an ambiguous connection attempt (see discussion of false negatives for the LBNL/ICSI datasets within this section). The

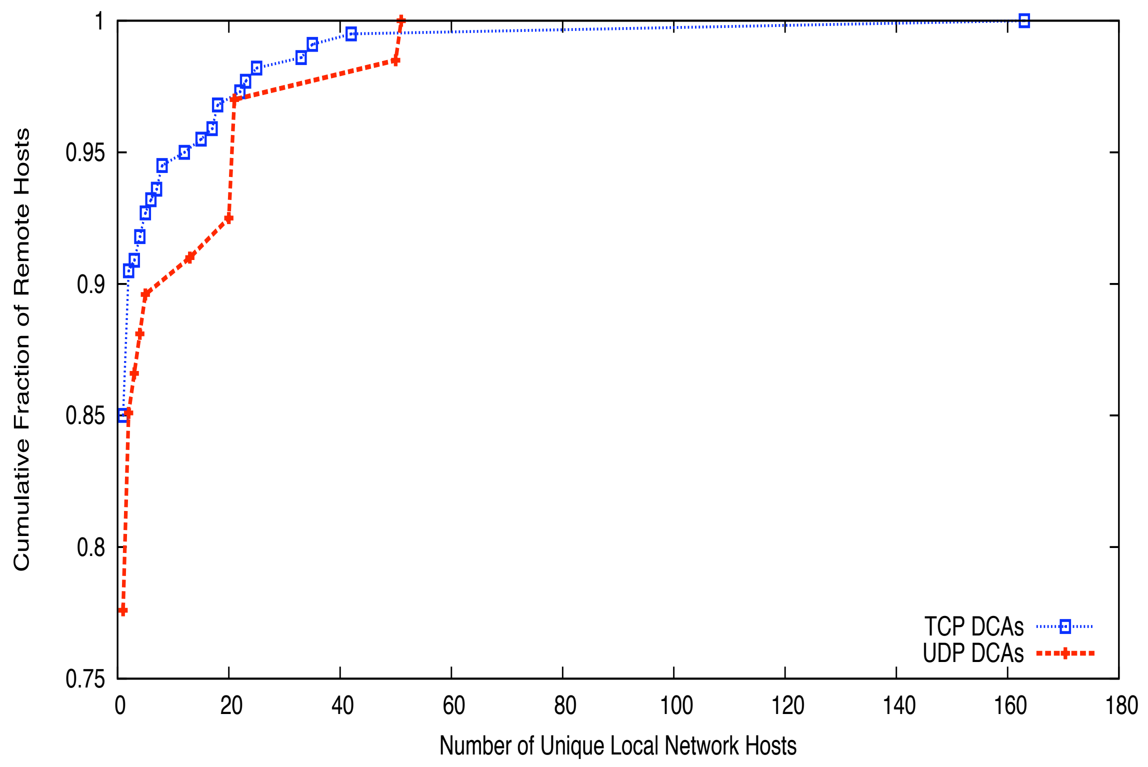


Figure 5.10: Fraction of Remote Hosts Generating DCAs Against at Most x Local Hosts. LBNL 12-16 Dataset, Using Exposure Maps Technique.

TRW and modified TRW technique both detected a total of 16 scans. All three techniques did not produce any false positives or negatives. Table 5.11 shows 3 071 DCAs generated by 287 unique remote hosts.

False Negatives for the LBNL/ICSI Datasets. The exposure maps DCA heuristic detected all scans identified by the TRW algorithm and therefore generated no false negatives relative to the TRW output, for these datasets (i.e. LBNL/ICSI 12-15 and LBNL/ICSI 12-16); see Tables 5.5 and 5.7. The exposure maps DCA heuristic did generate two ambiguous connection attempts, one in LBNL/ICSI 12-15 dataset and one in the LBNL/ICSI 12-16 dataset. This was determined by comparing, after careful analysis, the output of any scans uniquely detected by the exposure maps technique.

Further analysis (using `tcpdump`) of the TCP connections that generated these ambiguous connection attempts revealed the following common characteristics:

Table 5.10: Scanning Detection Comparison LBNL/ICSI 2004-12-16 Dataset. The values $\theta_1 = .20$ and $\theta_0 = .80$ were used for the Bro TRW and modified TRW techniques and a value of 4 *uniq-localhosts-scanned* per observation time window was used for the exposure maps DCA heuristic.

Technique		
Exposure Maps	Declared Scans	17
	False Positives	0
	False Negatives	0
	Ambiguous Connection Attempts	1
	Benign Hosts	n/a
Bro TRW	TRW Scans	16
	False Positives	0
	False Negatives	0
	Ambiguous Connection Attempts	n/a
	Benign Hosts	38
Modified TRW	TRW Scans	16
	False Positives	0
	False Negatives	0
	Ambiguous Connection Attempts	n/a
	Benign Hosts	38

Table 5.11: Additional Exposure Maps Results for the LBNL/ICSI 2004-12-16 Dataset.

Exposure Maps	
# of NEM Entries (TCP)	2 695 (1 652)
TCP DCAs	2 401
UDP DCAs	670
Total DCAs (TCP&UDP)	3 071
Unique Remote Hosts Generating DCAs (TCP&UDP)	287

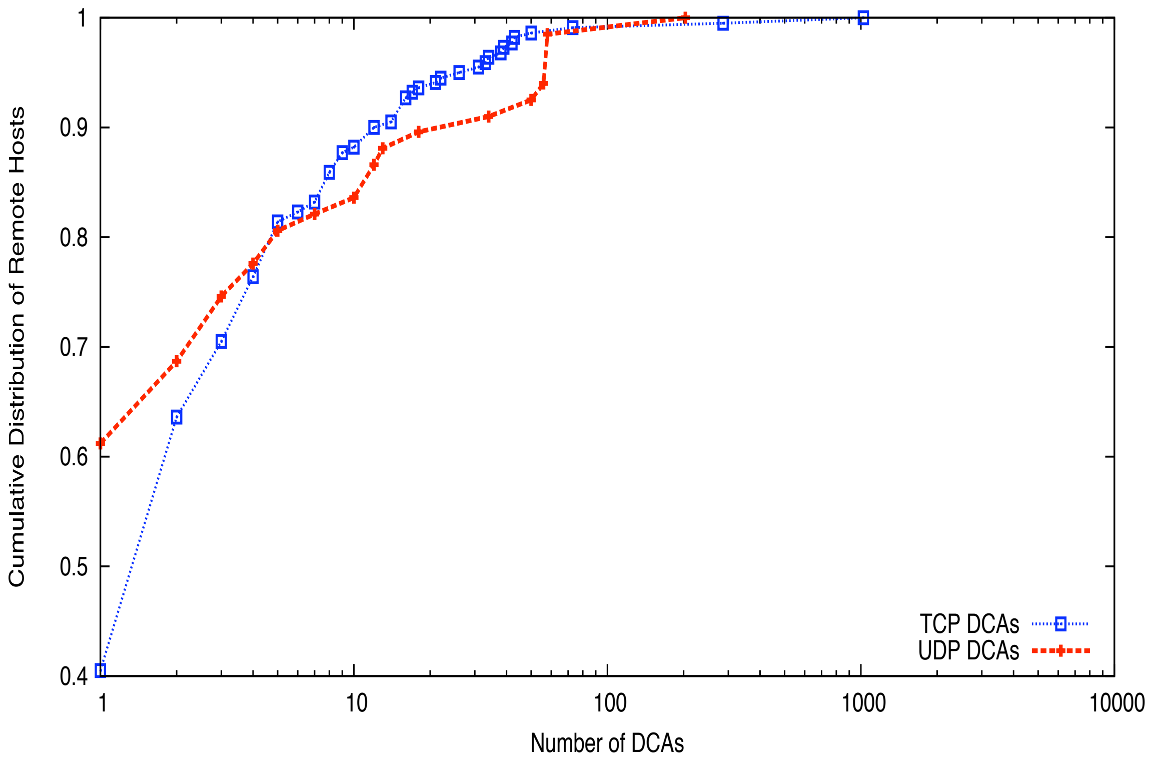


Figure 5.11: CDF of DCA Activity for LBNL/ICSI 12-16 Dataset (Using Exposure Maps Technique) – Number of DCAs.

- no initial three-way handshake: the set-up of the connection between the originator and responder of the connection was not present in the dataset.

- no graceful connection shut-down: the termination of the connection between the originator and responder in the connection was not present in the dataset.

- low remote host network interaction: the remote hosts (i.e. the scanners) associated with these ambiguous connection attempts had no other network activity present in the dataset other than the traffic that caused this unusual network activity.

- no destination service interaction: the destination hosts in the local network were never observed offering (i.e. no NEM entry for) the service being accessed by the remote hosts that generated these ambiguous connection attempts in these traces over the entire dataset.

Bro classified the connections that generated these ambiguous connection attempts using the OTH state indicating that the connection information it extracted from the network traffic to construct these flows was incomplete. Regardless, Bro makes an attempt to guess the roles of originator and responding system in the connection even when the three-way handshake that signifies the start of a TCP session is not present. It is not clear if these incomplete connections were an artifact introduced by anonymization, loss of packets during data collection due to the location of the collection equipment, or the result of ongoing long-term connections (see discussion in Section 4.2). In the exposure maps technique, as no three-way handshake for these connections was present in the dataset, the NEM would not have the IP/port entry of the responding system as an entry (i.e. the `connection_established` event would not be triggered which populates the NEM). However, this network traffic does trigger the Bro `new_connection` event and in the absence of an associated NEM entry for the destination IP and port/protocol, this activity is deemed a DCA. The TRW and modified TRW techniques did not flag this activity as a scan; they make use of connection states instead of connection events (see Section 3.1) allowing for a more fine-grained determination of whether a connection succeeded or failed and therefore did not consider this activity as failed connections.

Nonetheless, the TCP connection fragments that caused these ambiguous connection attempts are anomalous network activity (i.e. no connection set-up or tear-down) that may either be malicious in nature or simply an artifact of the method used to capture the datasets (see Section 4.2). Unfortunately, due to the fact that the datasets consisted only of anonymized headers, further analysis was not possible. Accordingly, although this network activity is suspicious, there is not enough information to support conclusion that the associated network traffic is a result of either benign or scanning activity.

Finally, the modified TRW technique also detected all scans identified by the original TRW algorithm. As previously discussed, scans directed against a port/IP combination contained in the NEM are not considered part of a scan but rather a connection attempt to a valid service; this might potentially then be a source of false negatives, and to claim otherwise (i.e. zero false negatives in general) would imply unknowable knowledge of the intent of the host requesting the connection.

False Positives for the LBNL/ICSI Datasets. As previously discussed, failed attempts to access a legitimate service would be classified as a DCA by the exposure maps technique as an attempt was made to connect to a host/port pair not listed in the NEM. Given that there is no way to measure the *intent* of a connection attempt, it is possible that if a remote host generated a number of these events (i.e. DCAs) against enough unique local hosts, this activity could be classified as a scan by the exposure maps DCA heuristic used in our comparison with TRW. As revealed in Figures 5.8 and 5.11, a significant number of remote hosts each generated a single DCA. Any remote host activity resulting in a single DCA would be below the minimum detection threshold for all three techniques used in this evaluation thus eliminating these as a possible source of false positives. The single false positive attributed to the TRW and modified TRW technique in the LBNL/ICSI 12-15 dataset was for the same scanning activity (i.e. the source and destination hosts as well as the associated network timestamps matched). Analysis of the network trace revealed that the remote host scanned a total of 3 unique systems not the 4 required by the threshold. We suspect that this phenomenon is a flaw in the implementation (i.e. software bug) and not the TRW algorithm.

5.1.2 Results: Advanced Scanning Detection

Attackers may disperse the scanning activity among several sources to reduce the overall scanning footprint in an effort to evade detection. To detect distributed scanning we propose classifying DCAs using the following criteria.

i. *DCAs and target destination ports.* The number of DCAs per unique source IP address is determined (in post-processing), through analysis of the DCA scanning activity log, over a configurable time interval (e.g. seconds, days). Similar numbers of DCAs from individual sources are grouped into clusters, which are then grouped by target destination ports. This final comparison reveals remote hosts with a similar scanning frequency (i.e. number of DCAs per unit time) and the same target service. We consider clusters of three or more remote hosts that target the same destination ports as a distributed scan;

Table 5.12: Three Detected Distributed Scans.

# of Scanners	Scanned Ports	# of Hosts Scanned	Follow-on Attack
3	80/TCP	62	No
14	22/TCP	62	Yes
9	25/TCP 53/TCP	62	Yes

Table 5.13: Scan/Attack Activity.

NEM Entry	Scan/Attack Entities	Scans or Attacks
10.0.0.1:25/TCP	5	5
10.0.0.2:22/TCP	40	4 545
10.0.0.2:80/TCP	17	120
10.0.0.2:443/TCP	4	9
10.0.0.3:22/TCP	40	10 601

the number of systems in a cluster is configurable.

ii. *Source IP proximity and target destination ports.* DCAs are first sorted by unique source IP address. Remote hosts in the same one-quarter class C subnet address range (i.e. /26) are grouped into a cluster. These clusters are then grouped by target destination ports. This reveals remote systems sharing a similar contiguous address space (which could indicate a single entity *Owns* the remote hosts) and target (i.e. service). Again, we consider clusters of three or more remote hosts as scanners that target the same destination ports as a distributed scan. Using these distributed scan heuristics, we detected three distributed scans in the CCSL network dataset (cf. Table 5.12).

The first consisted of three source IPs targeting port 80/TCP (HTTP). The scanning campaign was directed against the entire IP address range of the CCSL network (i.e. 62 systems). Once the scanning activity completed, no attacks were detected from these scanning sources. In fact, the only network activity exhibited by the systems participating in the distributed scanning campaign in the network trace was this specific distributed scan.

Table 5.14: Distributed Scan Characteristics.

DCAs
18:01:30 10.0.138.232 > 192.168.1.3.22
18:01:30 10.0.138.237 > 192.168.1.6.22
18:01:30 10.0.138.229 > 192.168.1.9.22
18:01:30 10.0.138.237 > 192.168.1.2.22
18:01:30 10.0.138.229 > 192.168.1.10.22
18:01:30 10.0.138.226 > 192.168.1.14.22
18:01:30 10.0.138.236 > 192.168.1.18.22
18:01:30 10.0.138.230 > 192.168.1.24.22
18:01:30 10.0.138.234 > 192.168.1.16.22
18:01:30 10.0.138.234 > 192.168.1.11.22
18:01:30 10.0.138.233 > 192.168.1.8.22
18:01:30 10.0.138.228 > 192.168.1.25.22
18:01:30 10.0.138.236 > 192.168.1.21.22
18:01:30 10.0.138.231 > 192.168.1.29.22

The second distributed scan consisted of 14 systems targeting port 22/TCP (SSH). The scan was also directed at the entire IP address range of the CCSL network. Two of the hosts in the CCSL network offer services on port 22/TCP. In contrast to the first distributed scan, two of the scanning systems attacked both systems in the CCSL network that offered the service (cf. Table 5.4). The number of DCAs from the 14 scanning systems ranged from 2 to 7. The network address space was scanned in non-sequential order and the entire scan lasted 1 second. Table 5.14 summarizes the characteristics of this distributed scan by showing the first 15 DCAs of the scanning campaign. During the entire campaign, the source IP address varies among the 14 scanning hosts with no occurrence of two DCAs arriving one directly after the other from the same source IP. Approximately 2 minutes after the scan concluded, all 14 of the systems that participated in the distributed scan attacked (with repeated login attempts) precisely the two systems in the network that offered SSH. Analysis of the network traces showed that the repeated SSH login attempts were all unsuccessful.

To visualize this specific distributed scan and subsequent attack, we make use of the Bro connection information stored in the `conn.log` and plotted it in a directed graph. Figure 5.12 shows the 14 distributed scanners (i.e. red oval nodes) attempting to make a connection to port 22/TCP on all the hosts within the CCSL network (i.e.

blue oval nodes labeled H1 to H62 that represent the 62 hosts in the local network address range). 60 of the nodes do not accept the incoming connection attempts (i.e. Bro connection state `REJ` in the red rectangle), however two nodes accept the incoming connection attempts (i.e. the Bro states `SF` and `S0` indicate the service is offered, the blue and purple rectangles).

Figure 5.13 shows this same activity plotted in a directed graph in terms of connections to the targeted port of the distributed scan (i.e. 22/TCP). The distributed scanners (red nodes) all made connection attempts to port 22/TCP (yellow node) on all 62 local network hosts. 60 of the hosts (white nodes) did not respond while two of the hosts (light blue nodes) did.

Finally, the third distributed scan detected consisted of 9 scanning systems targeting ports 53/TCP (DNS) and 25/TCP (SMTP). One of the hosts in the network offers services on port 25/TCP and two offer services on port 53/UDP (not 53/TCP) respectively. Again, all 62 hosts in the CCSL network were scanned with an attack immediately following on the system that offered port 25/TCP.

The distributed scanning detection heuristics described above illustrate how DCAs detected and recorded through exposure maps can be post-processed to detect sophisticated scanning activity. Other heuristics may be developed that use the raw output from exposure maps scanning logs to identify other types of simple or sophisticated scanning activity (e.g. slow scanning). For instance, as an example third heuristic, to detect slow scans to a particular service (i.e. port) one can use the timestamp feature from the recorded scan events. Some time-constrained set of detected DCAs is sorted by source IP address. Using the timestamp as a reference, scan intervals of less than 5 minutes from a particular source IP address to the same destination port are ignored. This heuristic would detect scans from a single host to the same destination port on multiple hosts with a scan interval of 5 minutes or greater.

5.1.3 Results: Active Response

Of the 813 remote hosts responsible for generating DCAs in the CCSL dataset, 66 launched a total of 15 280 DCAs intermingled with attacks (unsuccessful) against the network, e.g., repeated attempts to relay mail through the mail server, and attempted

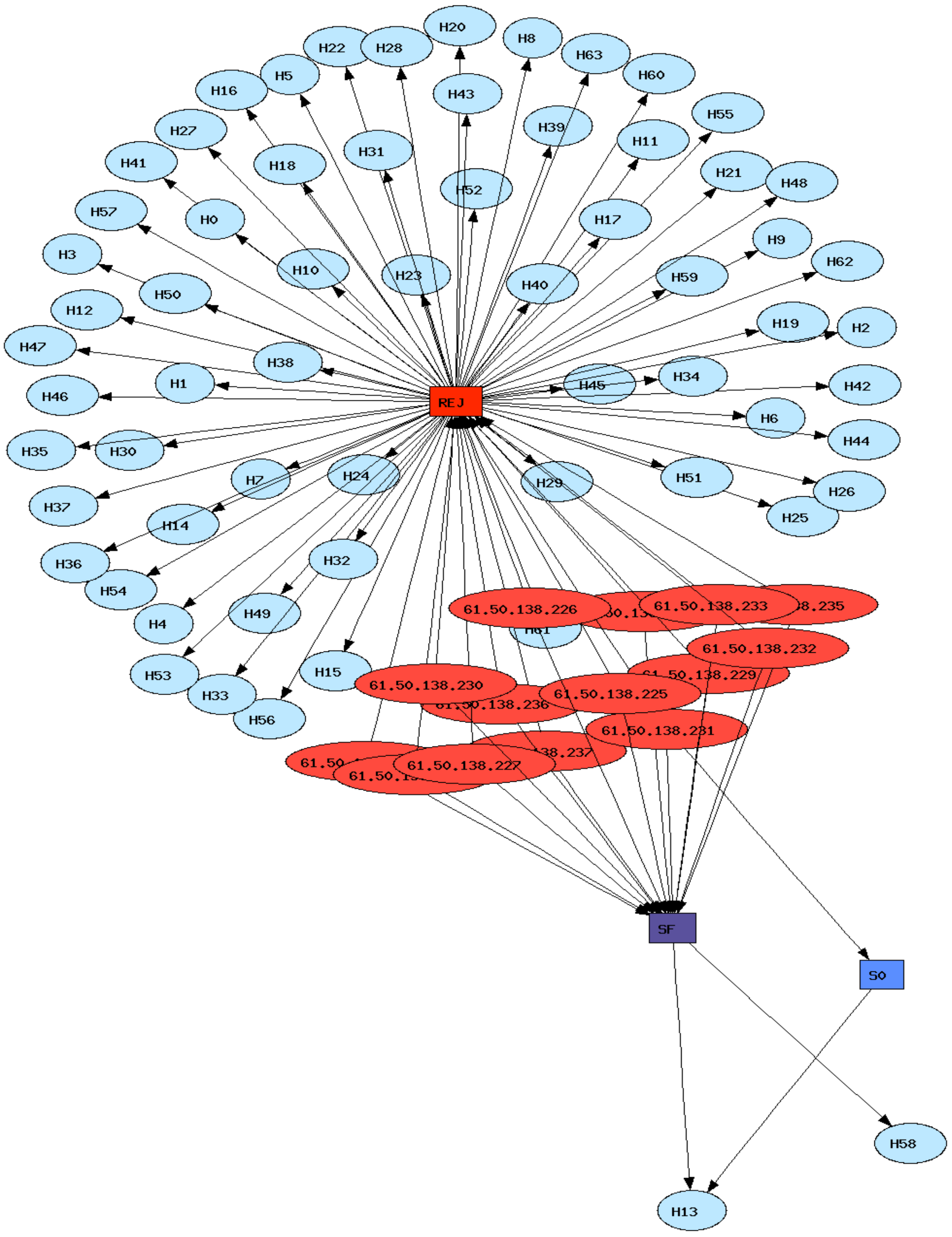


Figure 5.12: Distributed Scan of the SSH Service on CCSL Network IP Addresses.

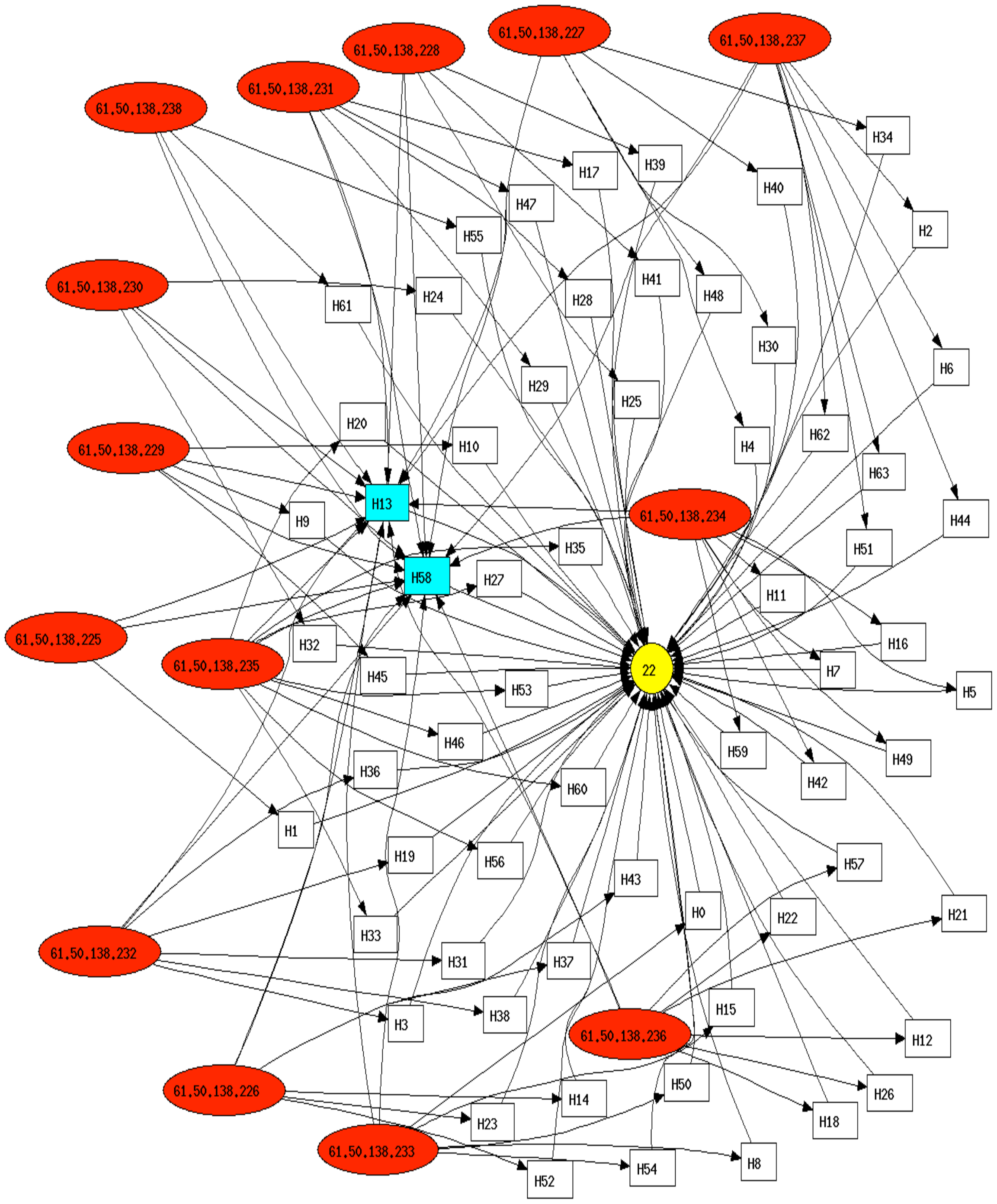


Figure 5.13: Distributed Scan of the SSH Service on Port 22/TCP of CCSL Network IP Addresses.

logins to an SSH service. Mail relaying is prohibited by the CCSL network's mail server and the responses from the mail server to the attacking system indicate that no relaying occurred; analysis of the network traces also showed that the repeated SSH login attempts were all unsuccessful. Some of these remote hosts scanned and attacked multiple services; this explains why the number of scan/attack entities in Table 5.13 is 106, while the actual number of unique IPs addresses was 66. With the exception of a single distributed scan, two characteristics of this activity occurred: (1) scanning was always the precursor to the actual attack, and (2) whenever a scan was directed against a service offered by the network (i.e. entry in the NEM), an attack followed once a response to the scan was sent. This "scan then attack" activity fits the profile of autorooter or worm activity as described in Section 2.2.1. The attacks were directed against four services offered by the network: SMTP, HTTP, SSL, and SSH.

Without the knowledge of what services are offered and in active use by the network, in a standard perimeter defense all 813 source IPs that generated DCAs over the four week period might need to be blocked at the router or firewall. The NEM provides up-to-date knowledge of the external interface of the network. The NEM, coupled with the technique of Section 4.1.2, allows a minimal set of only 66 remote hosts should be denied access to the network. This represents a 92% reduction in the number of dynamic updates to the network security ACLs.

5.1.4 Results: Exposure Profiles

Exposure profiles offer the ability to passively perform host discovery and identification. To determine how well exposure maps can be used to identify and group hosts with similar HEMs into exposure profiles, we tested this feature on the two LBNL/ICSI network traces. We classified all the HEMs using the profiles of Section 4.1.3; specific ports/protocol for the types of applications listed in two of the example profiles are listed in Table 5.15. The mapping from port number to network service is contained in Appendix A.2, Table A.2. These specific profiles were selected to demonstrate the feasibility of using HEMs to group their respective hosts into specific network profiles. Accordingly, the number of profiles and specific services included in each are configurable for different network environments.

Exposure profiles can be used to rapidly partition a NEM's hosts into subsets of

Table 5.15: Exposure Profiles Port Assignments.

Risk Level	Ports
Low	21/TCP, 22/TCP, 23/TCP, 25/TCP, 53/TCP, 53/UDP, 80/TCP, 110/TCP, 111/TCP, 111/UDP, 113/TCP, 119/TCP, 138/UDP, 143/TCP, 161/UDP, 427/UDP, 443/TCP, 515/TCP, 554/TCP, 993/TCP, 995/TCP, 1755/TCP, 1863/TCP, 5050/TCP, 5061/TCP, 7000/TCP
Medium	135/TCP, 445/TCP, 1433/TCP, 3389/TCP

Table 5.16: Exposure Profiles.

LBNL/ICSI Datasets			
Network Trace	# of HEMs	Low Risk	Medium Risk
12-15	1 741	1 370	316
12-16	1 410	1 077	434

the NEM, based on the entries in their HEM. For instance, Table 5.16 summarizes the number of HEMs (hosts) within two exposure profiles for the selected network traces from the LBNL/ICSI datasets. A HEM's placement in a profile in this example is determined by the *highest* risk service it offers; a HEM that contains entries 22/TCP, 80/TCP, and 1433/TCP would be included in the Medium Risk profile.

5.1.5 Results: Reconnaissance Activity Assessment (RAA)

Table 5.17 shows the CCSL NEM enhanced with application information that is transmitted by the active hosts in the CCSL network when a connection is made to an offered service (i.e. a NEM entry). The entries represent the externally visible interface (network and application layer) of the network.

Recall that Table 5.13 shows the scan/attack results from the CCSL dataset. Table 5.18 is a result of overlaying this information with the enhanced NEM. The enhanced NEM makes it possible to perform RAA and determine what specific application information from the network was sent to the detected scanning systems. For instance, the 14 distributed scanners that targeted port 22/TCP on all 62 hosts in the CCSL network (cf. Table 5.12) would have had at least the following information returned to them as a result of their scanning campaign:

Table 5.17: Enhanced CCSL NEM showing Application Information.

Host	Port	Application Information
10.0.0.1	22/TCP	`OpenSSH 3.8.1p1 (Protocol 2.0)`
	25/TCP	`Generic SMTP Exim (shakespeare.ccsl.carleton.ca)`
	631/TCP	`Unknown HTTP (HTTP/1.1)`
	993/TCP	`University of Washington IMAP daemon1`
10.0.0.2	22/TCP	`OpenSSH 3.8.1p1 (Protocol 2.0)`
	80/TCP	`Apache 2.0.54 (Debian GNU/Linux)`
	443/TCP	`Generic TLS 1.0 SSL`
10.0.0.3	22/TCP	`OpenSSH 3.8.1p1 (Protocol 2.0)`

Table 5.18: RAA for CCSL Dataset.

NEM Entry	Scan/Attack Entities	Scans or Attacks	Application Information
10.0.0.1:25	5	5	Generic SMTP Exim (shakespeare.ccsl.carleton.ca)
10.0.0.2:22	40	4 545	`OpenSSH 3.8.1p1 (Protocol 2.0)`
10.0.0.2:80	17	120	`Apache 2.0.54 (Debian GNU/Linux)`
10.0.0.2:443	4	9	`Generic TLS 1.0 SSL`
10.0.0.3:22	40	10 601	`OpenSSH 3.8.1p1 (Protocol 2.0)`

i. Host 10.0.0.1: port open – `OpenSSH 3.8.1p1 (Protocol 2.0)`.

ii. Host 10.0.0.3: port open – `OpenSSH 3.8.1p1 (Protocol 2.0)`.

iii. All remaining hosts in the network (10.0.0.2, 10.0.0.4-62): ports closed – no response.

A variety of `regex` expressions [52] are included in the `darkport-sig.sig` file to detect a number of popular applications. This file is easily extensible to accommodate new applications.

5.2 Scalability and Stability of Exposure Maps

The size of the NEM will be determined by numerous factors, the two most important being the number of distinct hosts using the monitored link (or network), and the variety of applications those hosts use. We tested exposure maps technique on the LBNL/ICSI network datasets to determine scalability in larger and more diverse environments; individual traces included tens of thousands of unique source and destination IP addresses.

The resource consumption of exposure maps includes system detection state and disk storage. The former refers to storage for the features extracted from network events (i.e. port/IP address for a NEM and port/IP address/associated application information for an enhanced NEM) that must be maintained at all times in main memory, that will provide, at wireline speed, the context to build and maintain the exposure maps as well as perform its various applications.[5] A number of techniques are used by other network-based scanning detection approaches to limit their use of allocated system resources (CPU cycles, main memory, disk storage). These include connection timeouts, reduction of monitoring windows, fixed sized memory buffers and analyzing only certain events/protocols all of which may have an adverse effect on the accuracy of the detection technique. The disk storage usage as a result of using the exposure maps technique will depend on the detected scanning activity, increasing with the number of DCAs recorded. We now discuss the system detection

[5]Depletion of this finite resource, due to traffic volume or an intentional DoS attack, can overload and defeat a detection system. Resilience to attack is discussed separately; see end of this section.

state and disk storage requirements for the various applications of exposure maps, and computational overhead.

Scanning Detection Scalability. For scanning detection with exposure maps, (1) the NEM (vetted or unvetted) must be constructed and maintained, and (2) DCAs must be written to disk in a DCA activity log file. To understand the amount of detection system state for (1), consider the largest NEM observed in the LBNL/ICSI 12-15 dataset, consisting of 3 604 entries. Each NEM entry contains 6 bytes: 4 for IP address and 2 for port. Thus, the total memory footprint for this NEM is $6 * 3\,604 \approx 22K$ bytes plus additional overhead for storage in a data structure (i.e. in our implementation the Bro `set` data structure). Thus, with an allocation of less than fifty Kilobytes in system detection state (i.e. main memory), we can perform scanning detection for this network. As new connection requests are received, a single lookup is performed on the NEM with the destination IP and port fields from the incoming request to determine if there is a match. The small amount of detection system state coupled with the minimal computational overhead required to determine if an incoming connection request matches a port/IP pair in the NEM (i.e. analogous to a single lookup in a hash table) make this technique suitable for use at wire speed even in large enterprise environments.

To estimate the disk storage required for DCAs in the DCA log file, we examined the LBNL/ICSI and CCSL datasets. The unoptimized 5-tuple and associated notification/alarm type (e.g. DCA, NewService – see Section 4.1.5) that represents each DCA, in character delimited ASCII, requires 146 bytes of storage. To store approximately one day of DCAs for the LBNL/ICSI 12-15 dataset would require 5.7 Gbytes; or for one month, 171 Gbytes.[6] In contrast, to store the 747 320 DCAs detected in the CCSL dataset (4 week period) in the DCA log file would take 104 Mbytes. Assuming the dataset represents an average level of DCAs, an entire year of DCAs for the CCSL network ($\approx$ 1.2 Gbytes) could be stored on a single DVD or USB key.

Long term event storage is useful for applying heuristics to detect sophisticated scanning activity (e.g. slow scanning) and scanning trend analysis. Part of our future work includes optimizing the way that DCAs are stored, to significantly reduce the

[6]A commodity external hard disk with this storage capacity costs $\approx$ \$50. The LBNL/ICSI 12-15 dataset had a local network that consisted of approximately 11 000 hosts.

disk storage required e.g., through the use of binary output, DCA aggregation, and compression.

Automated Response Scalability. The automated response application is more expensive on system detection state than the scanning detection application due to the scanners list (recall Section 4.1.2). Each entry in the scanners list requires an additional 4 bytes (plus additional overhead for storage in a data structure). As connection requests are received, an additional lookup is required (i.e. a check against both the NEM and the scanners list) to determine if the source IP address matches an entry in the scanners list. We detected 831 suspected scanners in the CCSL dataset. With an additional allocation of 10K or less in system detection state, we could enable the automated response application. The length of time an IP address determined to be a remote scanner stays on the scanners list is configurable.

Exposure Profiles Scalability. The exposure profile application is the least expensive in terms of system resources. To build exposure profiles, we need to construct and maintain the NEM as for the scanning detection application. The HEMs in the NEM are simply sorted and logically grouped by their respective ports into the respective profiles. There is no requirement to write any information to secondary storage.

Reconnaissance Activity Assessment (RAA). To perform RAA in the network an enhanced NEM is required. This is more expensive in terms of system detection state than a regular NEM (i.e. composed solely of IP/port entries). The additional information stored in each enhanced NEM entry consists of any application information extracted from the network in near-real time as a result of the `regex` expressions found in the `darkport-sig.sig` file. As, the `regex` expression performs pattern matching on network traffic in real-time (the same technique employed by misuse network intrusion detection systems), the greater the number of `regex` patterns to be matched or the faster the speed of the network link, the greater the potential for a negative effect on the performance of the system.

Stability. The time for a NEM to become stable (i.e. infrequent trans-darkport activity) will vary greatly depending on the environment in which it is used. In an enterprise network with a tight network security policy (e.g. government, finance, health care), we would expect the NEM to stabilize quickly and thus be suitable for use in a scanning detection technique. As noted in Section 5.1.1, in the CCSL network the NEM stabilized in 20 hours. In other environments, service usage may vary by day of the week. In a network environment with an open network security policy, an unvetted NEM may be used which will scale well but not stabilize as new hosts continually enter and leave the network (e.g. mobile users) and new applications and services are continually added to client systems (e.g. P2P file sharing, open proxies). An unvetted NEM is suitable to use in the core network of an ISP as the concept of network boundaries and universal network security policies are not applicable. Additionally, in these *fluid* network environments, exposure maps remain useful, e.g. as a tool to perform network discovery through the application of exposure profiles, as discussed in Sections 4.1.3 and 5.1.4.

DoS Attacks. A potentially serious attack on many scanning detection mechanisms is one that specifically targets the detection system. In this context, we review the general construction and maintenance of basic exposure maps, plus the four main applications considered (scanning detection, automated response, exposure profiles, and RAA).

The construction and maintenance of either basic or enhanced exposure maps (both vetted and unvetted) appears resilient to DoS attack. Incoming scans (bursts or sustained activity) do not increase an exposure map's size (i.e., the number of HEM entries), which reflects only the number of services offered by the corresponding host. Incoming scans do need to be passively monitored, and connection requests are checked for matches against the NEM; however, the processing required for this is minimal, and we would expect any problems caused by volume of requests to cause other elements of a network to fail, e.g., having adverse affect on core network devices such as routers, or firewalls. Similar to basic exposure maps, the exposure profile application appears resilient, as neither disk storage nor system detection state are adversely affected by attack; exposure profiles rely only on exposure maps to logically group system devices based on the services they offer.

In the scanning detection application, secondary storage may be adversely affected by a large botnet DoS effort, because detected scanning activity is recorded. For example, for a 100,000 system botnet executing a scanning campaign on a target network, three simultaneous scans by each bot would consume 41.8 Mbytes in the DCA activity log. A sustained scanning effort by such a botnet would exhaust disk storage in most networks. However, such an attack would also likely cause core network devices to fail as noted above.

The automated response application would experience the same impact on disk storage as the scanning detection application, plus system detection state would be consumed for source IP addresses added to the scanners list (as incoming connection requests to port/IP combinations outside the NEM result in new scanners list entries). A botnet of size 100,000 would consume more than 400Kbytes of (in a scanner list) system detection state; this state consumption does not increase after the first scan from each source IP address. The most successful attack would likely be an attacker intentionally trying to exhaust scanner list state by spoofing source IP addresses during a large scanning campaign; this could adversely affect the platform executing the automated response application.

The RAA application is resistant to DoS as well. As is the case with the basic NEM, incoming scans do not increase an enhanced exposure map's size which reflects only the number of services offered by the corresponding host. Although the enhanced NEM is constructed by `regex` pattern matching, only those connections that succeed need to be processed. The majority of scans do not succeed (see Section 5.1.3) and therefore result in half-open or rejected connections with no traffic content being sent between the server and client. Only in those instances where a scan succeeds will connection processing be required to perform pattern matching for the enhanced NEM (if used).

5.3 Further Discussion and Limitations

In order to evaluate the exposure maps technique, we used the datasets from two different network environments and performed a side-by-side comparison with (1) the TRW scanning detection algorithm [29] implemented as a Bro policy and (2) a modified TRW technique augmented with a NEM as a second check to determine the success or failure of new connections (described in Section 4.1). The evaluation included two test runs on the CCSL network dataset that varied the detection threshold of the TRW algorithm by changing the sensitivity of the detection technique based on the number of active hosts and services offered in the network to determine the effect on the false positive and negative rates. Specifically, two values of θ_1 were selected to reflect the probability that a scanner's connection attempt would succeed based on the host and service density in the network. All three techniques recorded no false negatives or positives over both test runs.

The comparison was limited as a result of both the implementation of the exposure maps technique and the parameters chosen for TRW in our tests (see evaluation limitation discussion in Section 5.1). Specifically, the limitation of the exposure maps algorithm that constrained our evaluation stems from the fact that, unlike TRW, successful connection attempts are not individually tracked by remote hosts and thus are not a factor in the identification of scanning activity. The exposure maps technique has no concept of a likelihood ratio nor are any attempts made to identify benign hosts based on connection behavior. Thus, depending on the heuristic used to process DCAs, a legitimate remote host that makes a few unsuccessful connection attempts to the local network could be misclassified by the exposure maps technique as a scanner thereby generating a false positive. During our evaluation, while we did not experience any false positives due to this type of connection behavior, this remains a potential source of false positives for the exposure maps technique in a direct comparison with TRW in different network environments or datasets.

The expected number of observations for TRW (per the parameters studied) to classify a remote host as benign in the CCSL network is $E[N|H_0] = 2.330$ when θ_1 is set to the host density value and drops to $E[N|H_0] = 1.115$ when θ_1 is set to the service density value (see Section 5.1). In fact, the Bro TRW implementation of TRW we used in our evaluation when testing θ_1 that reflects the service density of the CCSL network will classify a remote host as benign after the observation of only a single successful connection (in the absence of any failed connections). It could be argued that a single observation of a successful connection attempt is not enough evidence to conclusively support H_0. Specifically, the requirement of only a single successful connection attempt (in the absence of any failed connection attempts) for a remote host to be considered benign may erroneously lead to the misclassification that a scanner is benign.

For instance, a number of scans observed against the CCSL network were from remote systems sequentially scanning specific ports on every host in the network starting at the low order address in the subnet and working their way through the entire network address space in strictly ascending or descending order (e.g. 10.0.0.1, 10.0.0.2, 10.0.0.3 ...). In the scenario where a network is configured such that the low order addresses are populated with active hosts (a typical configuration), the scanners we detected performing sequential scanning would have been misclassified as benign generating a significant number of false negatives. Fortunately, for TRW in this specific instance, the three active hosts on the CCSL network are not assigned to the start of the network address block and thus sequential scanning activity was easily detected by the TRW technique even when the service density value for θ_1 was used. Regardless, for the CCSL network we recommend that when using TRW, θ_1 be set to the host density to minimize the possibility of scanners begin misclassified by the TRW technique as benign. This emphasizes the need to select the TRW parameters carefully to balance quick detection (the classification of a host as scanners or benign) with the objective of producing low false positives and negatives.

If successful and failed connection attempts are interspersed, a greater number of connection attempts observations are required by TRW before a remote host can be reclassified from undetermined to either benign or a scanner. One drawback of TRW is that it is an attribution-based network scanning detection technique: attribution of the remote hosts is required to assign each host a *score* based on the connection history with the local network for comparison with thresholds η_0 and η_1. Attribution presupposes that the identification of the root cause of scanning activity is possible. This assumption makes the technique unsuitable for detecting a growing number of sophisticated scanning techniques (e.g. distributed scanning).

The tests on the LBNL/ICSI datasets used the TRW parameter values as described by Jung [28]. The exposure maps technique (using our exposure maps DCA heuristic) had no false positives or negatives recorded during the tests but did identify 2 ambiguous connection attempts. The TRW and modified TRW technique each had no false negatives and 1 false positive respectively. The ambiguous connection attempts detected by the exposure maps technique were caused by incomplete TCP connections most likely caused as an artifact of the dataset used in the evaluation (Section 4.2). Although we do not consider this network activity definitive proof that a scan has occurred, we do believe that such anomalous network activity is worth identifying as suspicious. Accordingly, we did not consider this a false positive for the exposure maps technique nor a false negative for the TRW and modified implementations. The TRW algorithm uses failed connection attempts to determine if a host is performing scanning activity and these incomplete TCP fragments were interpreted by the TRW algorithm as the a result of a successful connection rather than a failed connection attempt. The inability of the TRW and modified TRW technique to detect these ambiguous connection attempts can be attributed to its implementation in Bro rather than the conceptual technique. The TRW and modified TRW technique requires a correct determination that a new connection has either succeeded or failed in order to determine if scanning activity is occurring. The specific Bro events these techniques utilized to determined connection success or failure determined these TCP fragments were part of a successful connection. If the Bro implementation was modified to incorporate other Bro events or additional logic was used to interpret these types of incomplete TCP fragments as anomalous (i.e. not part of a successful connection) then this activity would be detected.

The exposure maps technique has the benefit of using its knowledge of the network services offered by individual hosts in order to classify a new connection as legitimate or a DCA. The exposure maps technique interpreted these incomplete TCP fragments as DCAs. Subsequent in-depth analysis of the dataset for this network traffic is inconclusive as there is not enough evidence to determine if these connections were caused by scanning activity, an artifact of the data collection process, or simply benign traffic. Although the exposure maps technique offered equivalent performance in terms of the false positive and negative rates, it has the benefit of offering the possibility, through the passive recording of all the services offered by the network (i.e.

NEM), of providing additional capabilities beyond scanning detection, as discussed in Section 5.4.

The modified TRW technique makes use of the TRW algorithm as implemented in the `trw-impl.bro` policy script. It is augmented with a NEM (see Section 4.1.5) and relies on both the NEM and Bro generated connection states to determine if a connection either succeeded or failed. In order for a connection to be considered a failure, both the check against the NEM and the interpretation of the connection states associated with the TCP connection (`trw-impl.bro`) have to indicate a connection failure. Accordingly, the modified TRW technique had the same false positive and negative rates as the TRW technique. However, a benefit of the modified TRW technique is that the NEM acts as a connection oracle that obviates the need for the algorithm to wait for connection responses in order to determine successes or failures. This could dramatically improve the performance of the TRW algorithm in terms of speed of detection. The modified TRW technique also has the benefit of offering the possibility, through the passive recording of all the services offered by the network (i.e. through the NEM), of providing an *awareness* of active hosts, network darkspace, and darkports. This information could be used to enable the other capabilities the exposure maps technique provides (described above).

Scanning Activity Trends. Another application of exposure maps is as follows. As in Section 4.1.1, each connection attempt to a darkport is recorded in the DCA activity log. Over time, the scanning activity detected by exposure maps can be analyzed to determine specific scanning activity patterns or long-term trends. For instance, a sudden burst in scanning activity directed against a service offered by the network may prompt the network operator to confirm the patch level for the software associated with that service. A number of open source security sites could be consulted (e.g. CERT) to determine if the activity may be the result of an emerging exploit or zero-day vulnerability. In the event no suitable explanation is found, the network operator may choose to closely monitor activity to the hosts that offer this service until the scanning activity returns to normal levels.

Non-standard ports. One of the strengths of the basic exposure maps approach (i.e. the basic NEM not the enhanced NEM) is that it need only maintain very little state when operational. It need not inspect or decode the content of a TCP connection, but only to observe external connection attempts (i.e. SYN packets) and record the IP address and source port if there is a response (SYN-ACK). Thus, the exposure maps technique uses port numbers to identify the offered service. Although port numbers are a good indication of the type of service offered, users may choose to install services that use non-standard ports, e.g., an HTTP server using port 8080/TCP or 8000/TCP instead of port 80/TCP. Of course, use of non-standard ports may limit access as client systems must know the listening port number before accessing the service. This has the greatest potential impact on exposure profiles, which group systems according to the services they offer; a standard server application using a non-standard port may be misclassified into another profile. In the case of creating a vetted NEM for scanning detection, this issue is less of a concern; non-standard port usage should be detected after training when the NEM is vetted.

However, if an enhanced NEM is used, a determination can be made on which applications are running on specific ports in a HEM to detect non-standard port usage. For instance, a HEM that offers a service on port 8888/TCP associated with the application information `Apache 2.0.54 (Debian GNU/Linux)` would indicate that a web server was being hosted on a non-standard port. Of course, while the `darkport-sig.sig` file is extensible, it can only detect application information for which it has an a priori signature. Signatures can be developed to identify specific applications if they exhibit some form of unique network traffic. The exposure maps technique makes use of application banners to identify the applications running on specific hosts (see Section 5.1.5).

Anonymized Headers. Two of the network datasets used in our evaluation (i.e. the LBNL/ICSI 12-15 and LBNL/ICSI 12-16 traces) contained only packet headers, anonymized using `tcpmkpub` [43], which scrambles IP addresses to preserve privacy. It is important that all occurrences of a specific IP address are consistently mapped to a single address within a dataset, to allow meaningful analysis of the network traffic. Consistency over longer periods is advantageous for analysis but also makes it easier to defeat the anonymization and recover private information [12]. The LBNL/ICSI datasets were analyzed separately and thus their respective durations are short. Furthermore, the traces contained no payload information precluding their use for testing the RAA capability as well as the ability to conduct in-depth post-evaluation analysis on the scanning detection results (e.g., only a cursory false positive and negative analysis could be performed).

5.4 Summary

A primary advantage of exposure maps technique is that is efficient to implement – it requires the passive observation, recording, and maintenance of a list of the services offered by the hosts in a network. Incoming scans (bursts or sustained activity) do not increase an exposure map's size (i.e., the number of HEM entries), which reflects only the number of services offered by the corresponding host. This simplicity translates into a very efficient use of system detection state (i.e. main memory) and computational resources that easily scales for use in large enterprise and backbone networks.

During our evaluation, our implementation of the exposure maps scanning detection application performed as well, in a direct side-by-side comparison, with both the TRW and modified TRW scanning detection techniques. The exposure maps technique also offers the benefit of the following capabilities.

i. *Network-centric awareness.* The identification of active hosts, network darkspace, and darkports allowing network-centric context that increases the fidelity of its scanning detection.

ii. *Fine-grained automated response.* The ability to deny access only to those scanning systems that directly threaten hosts in the network.

iii. *Network service identification.* The ability to classify hosts into exposure profiles based on the services they offer.

iv. *Reconnaissance Activity Assessment (RAA).* A mechanism to perform to determine the specific application information obtained by an adversary as a result of a specific scanning campaign.

Chapter 6

Internal Network Scanning Detection Strategy 1: DNS-based

This chapter describes a new technique to detect internal network scanning activity (i.e. L2R) using the domain name system (DNS) address resolution protocol. DNS provides a mapping between numeric IP address and alphanumeric domain names. Address resolution is the process of finding an address (logical or physical) of a host in a network. We have observed that scanning activity within a network can be detected by observing suspected scanning systems failing to utilize the DNS protocol. Although this technique was designed for the detection of network scanning worms in mind, the scanning methodology employed by known network scanning tools [21, 67, 66] is the same (see Section 7.2.2) and therefore they can be detected using this technique. We describe how this technique can be used to detect internal hosts performing remote network scanning as well as present our evaluation results using this technique. This approach has been published in Whyte et al. [71].

6.1 DNS-based Scanning Detection: Approach

In random scanning, the use of a numeric IP address by the worm, instead of the qualified domain name of the system, obviates the need for a DNS query. New connections from the network that cannot be associated with any DNS activity are considered anomalous. If we can observe and correlate all locally generated DNS activity and new connection attempts within an enterprise network, we have the means to detect L2L inter-cell or L2R network scanning activity. The technique does not detect R2L or intra-cell (i.e. within the boundaries of a cell) network scanning activity.

However, this approach must take into account valid instances where no DNS query is required to access a particular system or resource. Our analysis of DNS activity within a network reveals two instances where this occurs. The first results from accessing distributed application and content delivery services. The HTTP protocol allows URLs consisting of numeric IP addresses to be embedded within the data payload of an HTTP packet. It is common practice for busy websites to maintain or outsource their content to larger centralized image servers to allow for better web page retrieval performance. When a user accesses a website to retrieve a webpage, they may be retrieving the requested material from several geographically separated servers. It is not uncommon for the web page content to include an IP address of a centralized image server that the browser uses to retrieve an image or media file. In this instance, the browser uses this numeric IP address to retrieve the image and does not require a DNS resource record. Instead of having to perform a DNS request for the object, the numeric IP address is provided to the browser in the content of the web page. We consider this a valid connection attempt incidentally obtained by a previous DNS query.

The second instance includes those servers and services that are simply not accessed with DNS. An application may have the numeric IP addresses of systems it needs to access embedded in its configuration file. A user may specify connections to a server by entering an IP address from memory at a command line. In these instances, the application or user has a priori knowledge of the IP address of the server they wish to access. This can include but is not limited to network server communications, remote administration tools, and certain peer to peer (P2P) applications. HTTP, certain applications, and users are all legitimate sources of numeric IP addresses that can enable access to services and systems. Legitimate use of numeric IP addresses by applications and users can be identified and added to a whitelist for exclusion from the detection algorithm. Taking these exceptions into consideration (see Whitelists in Section 6.1.1), we consider any system that tries to access another system without receiving a valid DNS response as undertaking network scanning.

6.1.1 High-Level System Design

Our software system design uses the *libpcap* [2] library and is comprised of two logical components: the PPE and DNSCE. The Packet Processing Engine (PPE) is responsible for extracting the relevant features from the live network activity or saved network trace files. The DNS correlation engine (DNSCE) maintains in state all relevant DNS information, a whitelist, and numeric IP addresses embedded in HTTP packets extracted by the PPE. This information is used to verify both outgoing TCP connections and UDP datagrams. In this context, verifying means ensuring that the destination IP address of an outgoing TCP connection or UDP datagram can be attributed to either a DNS query, an HTTP packet, or an entry in the whitelist. The software can process either live network traffic or saved network traces in the *pcap* [2] file format. To detect L2R worm propagation, the software system must be deployed at all external network egress/ingress points. To detect worm propagation between network cells, a system would need to be deployed in each cell at the internal ingress/egress points.

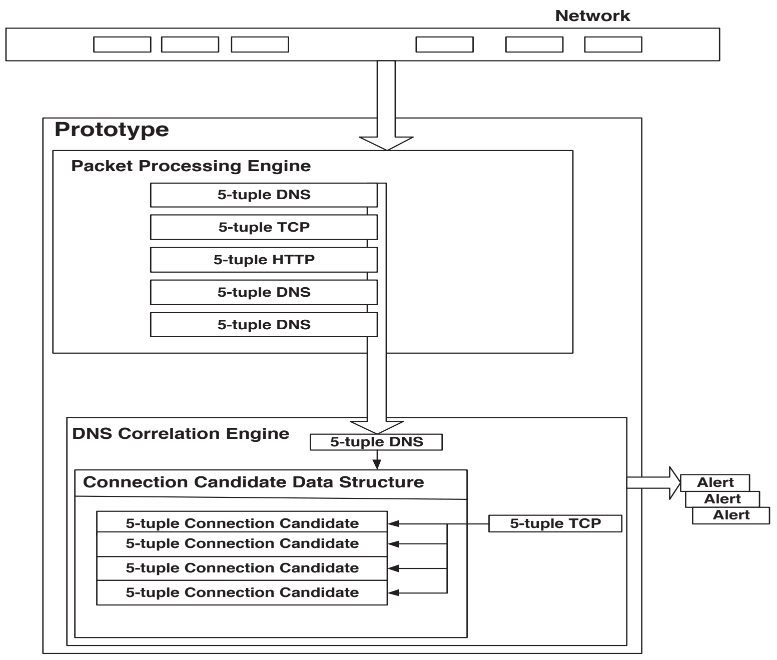


Figure 6.1: High-level System Design.

Figure 6.1 reveals the high-level design of the prototype. In this example, the PPE extracts the relevant features from live network activity and bundles these into data tokens. The data tokens are comprised of the appropriate 5-tuple of features based on the protocol extracted. These tokens are consumed by the DNSCE. The DNSCE uses the tokens to maintain a list of destination IP addresses it deems *valid* and checks any new connection attempts from within the enterprise network against this list. The DNSCE will generate an alert if it determines the new connection is being initiated to a destination IP that is not contained in its list (see Figure 6.2).

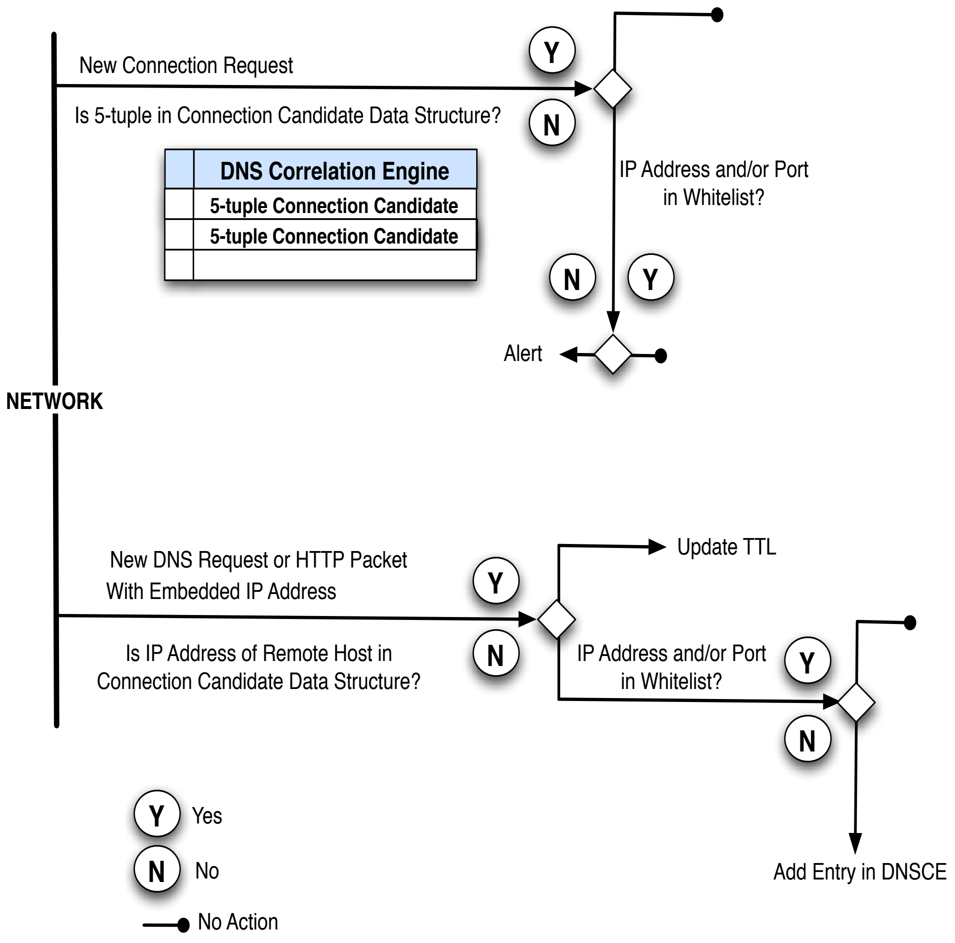

New Connection Request
Is 5-tuple in Connection Candidate Data Structure?
Y
N
DNS Correlation Engine
5-tuple Connection Candidate
5-tuple Connection Candidate
IP Address and/or Port in Whitelist?
N
Y
Alert
NETWORK
New DNS Request or HTTP Packet With Embedded IP Address
Is IP Address of Remote Host in Connection Candidate Data Structure?
Y
N
Update TTL
IP Address and/or Port in Whitelist?
Y
N
Add Entry in DNSCE
Y Yes
N No
No Action

Figure 6.2: DNS-based Detection Logic.

Packet Processing Engine. The PPE is responsible for processing packets of interest from *pcap* files or live off the network and extracts a variety of information from several protocols. Specifically, the software must extract relevant features from new connection attempts, embedded IP addresses within HTTP packets, and all DNS activity occurring within the network cell.

In order to discover new TCP connection attempts, all TCP packets with the SYN flag set are examined. TCP packets with the SYN only flag set indicate the start of the three-way handshake that signifies a new connection attempt. UDP is connectionless and does not have the concept of a session. Each UDP packet is treated as a discrete event and thus a potential new connection. Feature extraction for either new TCP connections or non-DNS UDP datagrams includes the 5-tuple of source IP, source port, destination IP, destination port, and timestamp.

Packets that contain a source port of 80 or 8080 are captured and categorized as HTTP packets. All HTTP packets are decoded and the payload inspected for any embedded IP addresses. Any IP addresses discovered in the payload are extracted along with the previously defined 5-tuple.

DNS A records are generated when systems within the network wish to contact systems in other cells or external to the network. Any DNS requests originating from the network cells and any DNS replies coming into the network cells are extracted and decoded. Feature extraction for DNS datagrams includes the 5-tuple of DNS source IP, DNS source port, TTL, domain name, and resolved IPv4 address.

DNS Correlation Engine. The DNS correlation engine (DNSCE) is responsible for processing information passed by the PPE. The two major functions of the DNSCE are: (1) to create and maintain a data structure of IP addresses and associated features that are considered valid connection candidates; and (2) to validate all new TCP and UDP connection attempts between cells or to remote systems against the connection candidate data structure. A valid connection candidate data structure is produced by processing DNS A records, embedded IP addresses in HTTP packets, and the whitelist.

Connection Candidate Data Structure. All DNS A resource record 5-tuples are parsed and added to the connection candidate data structure. The TTL from each 5-tuple is used just as it is in the cache of a DNS server. Once the TTL expires, the resource record is purged from the DNSCE's connection candidate list. Although DNS activity provides the majority of IP addresses to the connection candidate data structure, numeric IP addresses within HTTP packets must also be considered.

As previously discussed, numeric IP addresses are regularly embedded within HTTP packets. All HTTP 5-tuples are parsed and added to the connection candidate data structure. Unlike an IP address provided by DNS A records, these IP addresses do not have an associated TTL that can be used to discard the IP address entry from the connection candidate data structure. We can assume that a DNS query and response had to occur in order for the web site to be initially accessed. Therefore, we can use the TTL of the DNS A record of the original request as the TTL for the embedded IP address. All IP addresses harvested from HTTP decoding are then are maintained in state. That is, the assigned TTL values are respected and these addresses are valid only as long as the TTL has not expired.

Whitelists. To address those client applications that legitimately do not rely on DNS, a whitelist is generated. A whitelist provides a list of IP addresses and port combinations that are exempt from the detection algorithm. For example, in most networks there are systems that regularly communicate with one another by using IP addresses specified in configuration files rather than fetched in DNS records. Furthermore, specific applications and users (see further discussions below) may also use numeric IP addresses instead of DNS to access services or communicate with other systems.

In practice, internal network server communications are either well-known or easily discovered. If a hard-coded IP address is contained in a network configuration parameter or file, it is easily confirmed. These server interactions can be modeled and the appropriate IP address port combination added offline to the whitelist for exclusion. However, in the case of users, the use of numeric IP addresses may be more pervasive and more unpredictable. There are two cases worth discussion. In organizations which impose restrictive network security policies, end users are restricted to using a finite list of well known services deemed permissible in the security policies. For instance, it may be permissible to access FTP and Telnet servers using numeric IP addresses. To accommodate this, the list of frequently accessed FTP servers IP addresses could be added to the whitelist. Alternatively, so as not to weaken the security posture of the network, in such environments (e.g. financial and government) where an organization has tight control over its employees, users could be told to enter in domain names instead of the IP address. The second case, which may be more problematic for whitelists, involves end users which enjoy unrestricted or open network security policies. In this case, the number of whitelisted protocols may limit the effectiveness of the detector.

The whitelist is granular enough to exempt not only specific IP addresses but also provide for IP address and port pairing. For instance, it is possible to specify that a communication must contain the correct source and destination IP addresses as well as the correct destination port in order to match the applicable whitelist entry. Over time this list will need to be updated in order to reflect changes to the network, user activity, and new technology. The more open a network security policy, the greater the amount of effort required to maintain the whitelist.

New Connection Validation. The PPE only extracts the relevant information from a single TCP packet for each new TCP connection attempt it detects. This includes TCP SYN packets addressed to systems outside the cell the prototype is monitoring. Once a new TCP connection attempt is detected, the IP destination address is compared with the addresses listed in the connection candidate data structure. If the address is not found and it does not match an entry in the whitelist, the connection is considered to be anomalous and an alert is generated.

UDP datagrams are regarded as discrete events. The PPE extracts the relevant information from the UDP datagrams and passes this information to the DNSCE. Once a new UDP datagram is detected, the IP destination address is processed as described in the previous paragraph.

Alerts. An alert is generated when a connection attempt to a system in another cell or remote system is detected for which there is no associated entry in the connection candidate data structure. Multiple connection attempts between the same two systems within a specified time window are regarded as a single alert. This alert grouping reduces the number of alerts generated without reducing the relevant warning information to the operator. It is not unusual for a new TCP session to require a number of connection attempts before an actual connection can be established. Systems may be busy, unable or simply unwilling to establish a session. If a separate alert were generated for each unsuccessful connection attempt, a single communication between two systems may generate several alerts.

In regards to UDP, the decision to consider each UDP datagram as a possible new connection could result in numerous alerts that could quickly overwhelm an operator.

The important intelligence from these alerts is the identification of the potentially infected system and the intended victim. The fact that it took the worm multiple connection attempts or datagrams to infect the system does not aid in our propagation detection. The timestamp from the first TCP SYN packet or UDP datagram that generated an alert is used as the timestamp for the alert. The alert contains the time the activity was detected, protocol, source and destination IP address and source and destination port number.

6.2 Prototype Evaluation

6.2.1 Data Set for Prototype Evaluation

To validate our DNS-based detection approach, we developed and tested a fully functional software prototype. The software was installed on a commodity PC with a Linux operating system and a 10/100 network interface card. The prototype implements all features discussed in Section 6.1.1. To conduct our evaluation, one week of network traffic was collected at a firewall in front of one of our university's research labs (i.e. the Carleton Computer Security Lab – CCSL). A Linux system using *tcpdump* was connected to a tap in front of the firewall to collect and archive the network traces. We monitored both incoming and outgoing network traffic to the lab. The CCSL router is connected to the university's Internet accessible Class B network. The CCSL network consists of a one quarter Class C network (i.e. 62) of Internet reachable IPv4 addresses. Our evaluation did not include a side-by-side comparison of this technique with another L2R scanning detection technique as, to the best of our knowledge, no comparable publicly available L2R scanning detection technique exists.

The CCSL network contains one authoritative DNS server that all internal systems in the network are configured to use. The CCSL network's DNS server has entries associated with the CCSL network's mail server, web server, and Kerberos server. The firewall does not permit any inbound connections to client systems unless they were first established by an internal system. All systems within the CCSL network can access the Internet directly through the firewall, which is the sole egress/ingress point for the network. Using the cell definition previously described, the CCSL network can be considered one cell in the university's enterprise network. The CCSL network analysis allowed us to test the prototype's ability to detect L2R worm propagation.

During the course of our network traffic collection in front of the CCSL network firewall, network traffic from a separate internal university network was also captured. We will refer to this network as the Internal Departmental Network (IDN). The IDN has its own authoritative DNS server that all its internal systems are configured to use. The IDN can be considered another cell in the university's enterprise network. This incidental collection provided us with the opportunity to perform additional analysis. In addition to running the prototype against the CCSL network traces, we ran the prototype against a filtered version of the IDN network traces. To address privacy concerns, we restricted our inspection of the IDN's network traces to those packets that contained either a source or destination address that matched a CCSL network IP address. The IDN analysis allowed us to test the prototype's ability to detect worm propagation between cells.

At the start of our analysis, we flushed the CCSL network DNS server's cache. This ensured that any new connections from CCSL systems would result in an external DNS query to retrieve the appropriate A record instead of accessing the CCSL network's DNS server's cache. From our vantage at the network boundary, we are only able to detect DNS replies as they enter the CCSL network, not those generated internally from the DNS server's cache. The flushing of the CCSL network's DNS cache ensures that the DNSCE will contain the same DNS information as the CCSL network's DNS server. In our analysis, all IP addresses have been modified to keep the actual IP addresses anonymous. The university network's IP addresses are represented by the 192.168.0.0/16 IP address range.

Table 6.1: Network Data Set.

Network Protocol	Packet Count
TCP packets	5,969,266
TCP connections	18,634
ICMP packets	4,955
UDP packets	5,301,489
Other	805,604

Table 6.2: DNS Datagrams.

Date	Total Packets	DNS Request Datagrams	DNS Reply Datagrams
06-24-2004	2,101,243	6,485	6,264
06-25-2004	2,491,663	5,525	4,951
06-26-2004	847,687	1,192	658
06-27-2004	889,251	2,231	3,174
06-28-2004	1,339,283	5,225	4,752
06-29-2004	1,382,642	6,121	5,998
06-30-2004	1,081,451	4,973	4,164

Network traffic was collected for a seven-day period from June 24th to June 30th, 2004. The network traces are comprised of all network activity that reached the CCSL network's router from internal systems, systems in the IDN cell, and the Internet. During this period, over 5 million UDP packets were observed as well as almost 6 million TCP packets. A total of 18,634 individual TCP connections occurred. Table 6.1 provides the observed protocols and their respective quantities.

DNS is transported mainly over UDP. DNS zone transfers use the TCP protocol but it is a standard acceptable security practice to disallow this feature. Table 6.2 shows the number of DNS request and reply datagrams that were detected in the network traces. Overall, we observed that the total amount of DNS traffic is a small percentage of the total amount of network traffic. An individual DNS reply may contain multiple records. In fact, the 10,162 DNS replies we received in the network actually generated 99,994 individual DNS resource records.

6.2.2 CCSL Network Monitoring Analysis

The CCSL network deployment was used to test the prototype's ability to detect L2R worm propagation. Initially, we observed the network for a three-hour period the day prior to our data set to generate a whitelist. Section 6.3.1, Table 6.8 contains the seven entries that comprised the CCSL network's whitelist. In order for network activity to be identified as complying with the whitelist, the protocol, source IP, source port, destination IP, and destination port must all match. The first four entries consist of communications between specific servers. The fifth entry specifies a single server allowed to initiate connections with other systems on a specific port. Finally, the last two entries allow any system in the CCSL network to initiate connections to any other systems as long as they adhere to the particular port and protocol specified.

Over the course of the one week, a total of 52 alerts were generated by the prototype. Table 6.5 gives the alert breakdown by day. None of the alerts could be attributed to worm propagation but in contrast, see Section 6.2.3. This is not surprising since the CCSL network is a well-maintained hardened network administered by security-aware personnel. A full analysis of the true false positives generated by the prototype is given in section 6.2.4.

6.2.3 IDN Monitoring Analysis

The IDN deployment was used to test the prototype's ability to detect worm propagation between cells. Initially, we observed the network for a three-hour period the day prior to our data set to generate a whitelist. The whitelist for the IDN consisted of four entries (see Table 6.9 in Section 6.3.1). All of these entries consisted of allowed communications between specific servers. Over the one-week period, 74,610 alerts were generated as a result of worm propagation attempts from the IDN to the CCSL network. Table 6.3 contains the specific propagation attempts by date for each worm. Using the Symantec taxonomy for a naming convention, the three worms detected were: W32.Sasser, W32.Blaster, and a variant of W32.Gaobot. We estimate that these worms infect a total of 195 IDN hosts. Figure 6.2 illustrates the worm activity of the three worms over the entire analysis period.

In addition to the worm activity, the prototype detected other forms of scanning activity and as a result generated 191 alerts. Table 6.4 reveals the number and type of alerts generated. A full analysis of the false positives generated by the prototype is given in section 6.2.4.

6.2.4 Discussion of False Positives and Negatives

False Positives Results Analysis. 52 false alerts were generated from monitoring the CCSL network cell, and 191 false alerts from the IDN cell. Based on our analysis in the previous section, we have categorized these false positives as occurring due to:

Table 6.3: IDN Worm Activity.

Alerts			
Date	Sasser	Blaster	Gaobot
06-24-2004	25,052	1,104	3,299
06-25-2004	5,946	539	9,137
06-26-2004	8,894	721	761
06-27-2004	4,680	1,353	2,516
06-28-2004	739	1,085	21
06-29-2004	2,731	532	1,778
06-30-2004	1,383	1,680	659
Total	49,425	7,014	18,171

Infected Hosts			
Worm	Sasser	Blaster	Gaobot
Total	101	38	56

Table 6.4: Additional IDN Alerts.

Alerts	Activity
125	Optix Pro Trojan Horse scanning: port 3410 TCP
5	Random scanning: port 60510-60518 TCP
12	Ident/auth service: 113 TCP
49	Common Unix Printing System (CUPS): 631 TCP
Total Alerts: 191	

i. Authorized network communications that could be incorporated into a whitelist but were not in our prototype testing.

ii. Network configuration errors that could be eliminated with proper system administration.

iii. Suspicious scanning activity other than worm propagation.

iv. True false positives.

These are discussed in turn below.

Authorized network communications. For the purposes of our analysis, we initially allowed for a three hour training period to generate the whitelist. If this was extended to a few days, the whitelist could be augmented with additional entries, greatly reducing the instances of legitimate network activity generating false alerts.

Table 6.5: CCSL Network Alerts.

Date	# of Alerts	Known False Positives	True False Positives
06-24-2004	18	6 Internal CCSL Network, 3 Streaming Audio	9 HTTP
06-25-2004	20	4 Streaming Audio	16 HTTP
06-26-2004	1		1 HTTP
06-27-2004	6		6 HTTP
06-28-2004	1		1 HTTP
06-29-2004	4	2 Port 90 TCP	2 HTTP
06-30-2004	2	1 Port 90 TCP	1 HTTP
Total	52	16	36

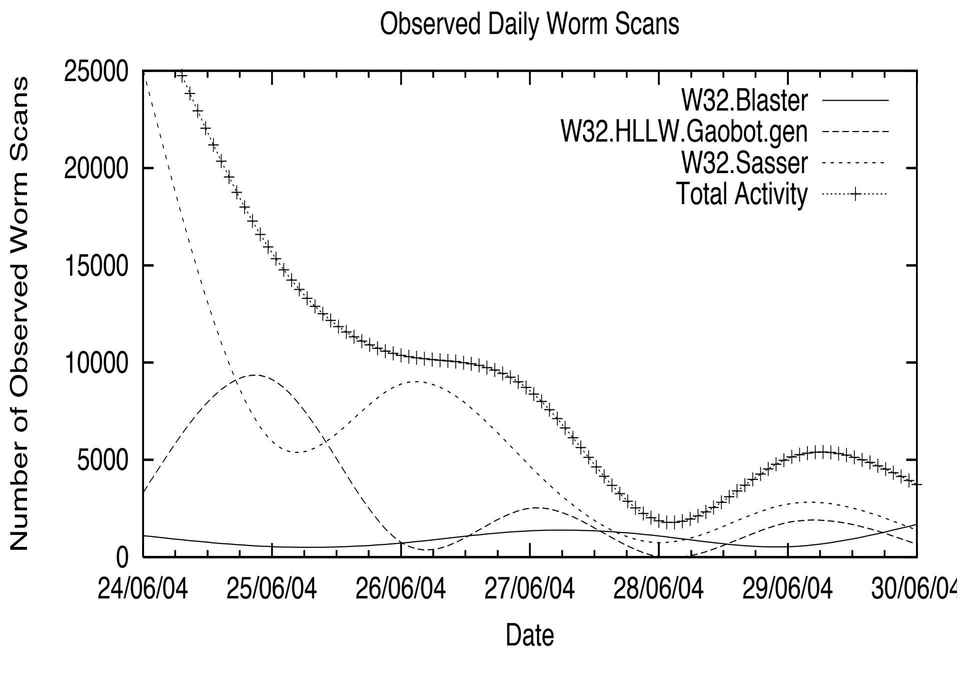


Figure 6.3: IDN Worm Activity.

Network configuration errors. 6 of the false positives were due to an isolated network configuration problem. Non-routable IP addresses passed through the firewall as a result of a router configuration error. Increasing the training period should also allow for sufficient time to detect any network configurations errors that may generate alerts.

Suspicious scanning activity. 125 alerts were generated as a result of an IDN system scanning for the Optix Pro Trojan horse [1] (i.e. port 3410 TCP). 5 alerts were generated as a result of an IDN system scanning for services listening on port numbers between the ranges of 60510 and 60518 TCP. Our preliminary version of the prototype software does not distinguish between scanning for the purposes of worm propagation or for some other activity. Although these alerts are considered false positives, they do warn an administrator that potentially malicious activity is occurring within the network cell. We discuss this in greater detail in Section 6.4.

True false positives. Those alerts that cannot be attributed to the previous three categories are considered *true* false positives.

After further analysis of the CCSL network monitoring results, we determined that 10 of the 52 alerts resulted from authorized network communications. 6 of the alerts resulted from a network configuration error. The remaining 36 alerts we classify as *true* false positives. With respect to the IDN monitoring results, we determined that 130 alerts were caused by non-worm related malicious activity and 61 alerts resulted from authorized network communications. None of the alerts were what we classify as *true* false positives. Table 6.6 summarizes the number and type of false positives generated from monitoring the CCSL network and IDN cells.

Table 6.6: False Positive Results Analysis.

	CCSL Network	IDN
Total Alerts	52	74,801
Worm Propagation Alerts	0	74,610
Pre-Analysis False Positives	52	191
Whitelist Inclusion	10	61
Network Configuration Errors	6	0
Suspicious Activity	0	130
True False Positives	36	0

Based on manual inspection of the network traces, we offer some insight into the cause of the 36 true false positives. The majority were caused by TCP connections initiated using a DNS resource record with a very low TTL and then not properly closed. It was a prototype design decision to detect a new TCP connection by simply observing a packet with the SYN flag set. The individual connections themselves were not tracked and maintained in state. Subsequently, we have observed HTTP connections that have been initiated using a DNS resource record with a TTL as low as 10 seconds. Several of these low TTL connections, all to the same web server, do not terminate properly. The client (i.e. inside the CCSL network) does not send the final ACK in the FIN ACK exchange. Instead, in some cases, the client sends a SYN to start a new session with the same server. This connection is initiated after the TTL has expired.

Approximately 60% (i.e. 22 alerts) of our true false positives were caused by this type of network traffic. Subsequent versions of our detection prototype could account for this network activity in two ways:

i. Enforce a minimum TTL: those TTL values lower than a minimum threshold (e.g. 600 seconds), could be given a default value (e.g. 600 seconds) in the DNSCE. According to our analysis, this would have reduced our true false positive count from 36 to 14 (i.e. 60% reduction). The increased probability of a false negative due to this arbitrary increase in TTL values would be negligible (see discussion later in this section).

ii. Require a second anomalous connection: we could modify our algorithm to generate an alert after two anomalous connections were observed from a system trying to connect to two separate systems within a finite time window. According to our analysis, this would have reduced our true false positives from 36 to 4. Requiring the observation of a second connection attempt would greatly reduce our false positives and only slightly degrade our detection precision (i.e. detect worm propagation after observing only two infection attempts).

Finally, although no UDP based alerts were generated during our analysis, we must comment on the false positive potential of persistent UDP connections. UDP datagrams are treated atomically by our prototype in that each datagram is verified against the connection candidate list. If the exchange of datagrams between the two systems is longer than the TTL of the DNS resource record that initially started the exchange, a false positive will be generated. This could become a concern if the TTL of the resource record is very low (i.e. typically the default TTL value is 1 day).

False Negatives Results Analysis. A false negative occurs when malicious activity occurs and no subsequent alert is generated. It was a design decision to monitor the network cell at the ingress/egress points, so that all new connections could be easily detected. Another consideration for this design decision was the fact that an end user can specify any DNS server they wish to use thus excluding the one administratively provided to them. As long as the network egress/ingress point is monitored, any external DNS queries can be detected and incorporated in the detection algorithm. However, by not monitoring the cell activity to the local DNS server, we will not be able to detect when the local systems contact the local DNS server. The prototype system maintains the DNS resource records in state respecting the TTL values for each record. If we detect an internal system starting a new connection to a remote system, the prototype checks the candidate connection list to determine if the connection is valid. In effect, we do not verify that the individual system has actually requested and received the DNS resource record, but rather that the resource record is available in the local DNS server. This is a subtle but important distinction.

Consider the scenario where an internal system becomes infected with a scanning worm. There exists the possibility that it may scan a system whose IP address is in the connection candidate data structure. That is, the intended victim was previously accessed by a system in the cell and the associated entry in the connection candidate data structure still exists. To determine the probability of this, we used the worm model discussed by Staniford et al. [7].

We assume that the worm targets victims at random over the entire IPv4 address space. Therefore, if r is the number of scans, a single host has a $N = \frac{1}{2^{32}}$ probability of being reached by the scan. If N is equal to the number of entries in the connection candidate data structure, the probability that a scan from the internal network will

Table 6.7: Probability of False Negatives due to Remote DNS Monitoring.

DNS Records	100 Infected Systems	200 Infected Systems	500 Infected Systems
500	5.821×10^{-6}	1.164×10^{-5}	2.328×10^{-5}
1000	1.164×10^{-5}	2.328×10^{-5}	4.656×10^{-5}
2000	2.328×10^{-5}	4.657×10^{-5}	9.313×10^{-5}
5000	5.821×10^{-5}	1.164×10^{-4}	2.328×10^{-4}
10000	1.164×10^{-4}	2.328×10^{-4}	4.656×10^{-4}

be directed at one of the data structures entries is

$$\beta = r\frac{N}{2^{32}}. \tag{6.1}$$

For example, with a 10,000 entry connection candidate list and a network that has 500 infected systems, if all of the systems began scanning at precisely the same time, the probability that after a single scan at least one of the scans would match an entry in the connection candidate list is only .04%. Table 6.7 contains probabilities, for various parameters, that a false negative will occur due to a single scan simultaneously occurring from each infected system targeting a previously cached IP entry in the connection candidate list.

6.3 Detection Circumvention and Current Limitations

In this section, we give an overview of possible ways our detection technique can be defeated and its current limitations.

6.3.1 Detection Circumvention

Any new worm detection algorithm will be the subject of scrutiny for both security researchers and malicious actors. The former seeks to validate and improve new ideas to realize improvements in overall network security countermeasures. The latter will devise ways to exploit weaknesses in the algorithm to circumvent detection. One method a worm writer could employ to evade detection from our approach is to have the worm do a preemptive valid DNS query before each infection attempt (i.e. scan). However, performing valid DNS queries before every infection attempt would have negative consequences for an attacker, e.g. it would:

Table 6.8: CCSL Network Whitelist.

#	Activity	Whitelist Entry	Reason
1	IMAP	192.168.1.33:993 - 192.168.200.50:993 TCP	Mail server
2	IMAP	192.168.1.25:993 - 192.168.200.50:993 TCP	Mail server
3	NTP	192.168.1.12:123 - 192.168.200.2:123 UDP	Server clock synch
4	NTP	192.168.1.12:123 - 192.168.200.1:123 UDP	Server clock synch
5	IMAP	192.168.1.5:993 TCP	Mail server
6	FTP	192.168.1.0/191:21 TCP	FTP sessions
7	SSH	192.168.1.0/191:22 TCP	SSH sessions

i. Require worms writers to adopt a new infection paradigm to randomly generate valid domain names instead of numeric IP addresses for targeting.

ii. Slow propagation as worms perform DNS queries in order to spread.

iii. Increase DNS activity as every infection attempt will pose a significant and noticeable impact on the DNS server.

iv. Reduce the number of *reachable* vulnerable systems because not all systems (e.g. home users, client systems in an enterprise networks) have qualified domain names being simply clients that do not offer any services. These client systems could comprise a significant portion of the susceptible Internet population depending on the exploitable vulnerability (e.g. Windows XP buffer overflow).

One of the limitations of the detection technique is that it cannot detect intra-cell and R2L attacks. A skilled attacker could use these limitations in concert to remain undetected while infecting the network. For example, a R2L attack is launched against the network and the worm infects a single system within a cell. Using the network information obtained from the infected system (e.g. netmask, broadcast domain), the worm can limit its scans to within the network cell. In parallel, another R2L attack could be executed against a system within another network cell and the process repeats.

Table 6.9: IDN Whitelist.

#	Activity	Whitelist Entry	Reason
1	IMAP	192.168.200.50:993 - 192.168.1.33:993 TCP	Mail server
2	IMAP	192.168.200.50:993 - 192.168.1.25:993 TCP	Mail server
3	NTP	192.168.200.2:123 - 192.168.1.12:123 UDP	Sever clock synch
4	NTP	192.168.200.1:123 - 192.168.1.12:123 UDP	Server clock synch

6.3.2 Current Detection Limitations

Our approach has limitations. As discussed in Section 6.1, it cannot detect R2L or intra-cell worm propagation. Automated attack tool activity (e.g. auto-rooters, network scanners, Trojan horses scans, etc.) will be detected but categorized as worm propagation.

Although it depends on the implementation, a worm that targets DNS servers may introduce irregularities that could limit the detection technique. Our detection technique also assumes that all applications honor TTL values; this may not be the case for all applications. Topological, metaserver, and passive worms may not trigger the detector depending on the behavior of the host programs [69]. Additionally, worm propagation via email/network share traversal will not be detected.

Finally, the use of whitelists in certain network environments could constrain the detection technique, as discussed in Section 6.1.1. Whitelists are used to exempt specific network activities and systems from the detection algorithm to reduce the occurrence of false positives. Tables 6.8 and 6.9 contain the whitelist entries for the CCSL and IDN networks respectively. These exemptions can be applied to the entire network (e.g. Activity 6, Table 6.8) or just to specific systems (e.g. Activity 5, Table 6.8). During our analysis of both networks, the following protocols comprised the respective whitelists:

i. Internet Message Access Protocol (IMAP): IMAP is a TCP based protocol that allows a client to remotely access email from a server [13].

ii. Network Time Protocol (NTP): NTP is a UDP based protocol used to synchronize computer clocks over a network [35].

iii. File Transfer Protocol (FTP): FTP is a TCP based protocol used to remotely exchange files [49].

iv. Secure Shell Protocol (SSH): SSH is a TCP based protocol used to encrypt a data stream to eliminate eavesdropping [80].

The CCSL and IDN networks have a strict security policy that restricts the type of services allowed within the network. Therefore, the whitelist entries for these networks were limited to a few systems and protocols. In a network with a very open security policy, the whitelist may become so large that maintenance becomes an issue and the detection algorithm either exempts too much network activity or creates too many false negatives to be useful. In this scenario, we believe that the DNS detection technique would best be used in conjunction with other detection techniques. For instance, the scan detection and suppression algorithm developed by Weaver et al. [70] could use our detection technique as another detection signal. In this scenario, our detection technique could provide a means to assign connections that did not use DNS a greater anomaly score than those that used DNS.

6.4 Extended Applications

Our DNS-based detection approach could be applied to five additional areas, which we believe warrant future investigation: (1) automated attack tool detection; (2) R2L worm propagation detection; (3) covert channel and remote access Trojan (RAT) detection; (4) mass mailing worm detection; and (5) integration with other anomaly based detectors. We discuss these in turn.

Automated Attack Tools. Automated attack tools share the same exploit methodology as scanning worms. Their goal is to rapidly identify and compromise as many systems as possible. A typical configuration parameter for automated attack tools is a range of numerical IP addresses that they use as their target information. The faster they scan, the faster they can compromise vulnerable systems.

Our DNS-based scanning worm detection technique can be used to detect automated attack tools. As part of our analysis during the prototype testing, we determined that 130 false positives were attributed to scanning for vulnerable services and previously installed malicious software (i.e. the Optix Trojan).

R2L Worm Propagation. R2L worm propagation refers to worm propagation attempts that originate from outside the enterprise network boundary. Our detection technique relies on observing DNS activity and new connection attempts from systems within the enterprise network. As we can observe all DNS activity initiated by internal systems, it is easily adapted to correlate this activity with new connections.

We believe that it would be possible to observe and correlate all DNS requests and new connection attempts initiated from remote systems. To determine the precision of the detection algorithm, requires further investigation of the difficulty of correlating DNS server requests with individual system connection requests.

Covert Channels. Covert channels are used to provide a communications method that violates the security policy of the system or network. Once a system has been compromised, an attacker typically requires some means to access the system to either exfiltrate data or maintain command and control. RATs typically use covert channels to communicate with their respective controllers outside the network. Covert channels are often created through software that can tunnel communications through well known and security policy sanctioned protocols in the network. For instance, several publicly available tools allow a user to tunnel data through the HTTP protocol.

Often an attacker uses a previously compromised system to attack other systems to evade detection. A large percentage of Internet systems (e.g. home users) do not have a fully qualified domain name associated with their IP address. Furthermore, it would not fit the profile of being covert if a compromised system had to perform a DNS query to identify the system that had surreptitious control of it. In this scenario, our DNS-based detection approach would be able to detect covert channel communications.

Mass-mailing Worms. Mass-mailing worms infect systems by sending infected email messages. The worm payload is typically contained within an email attachment. As part of the installed code base, these worms often contain their own Simple Mail Transfer Protocol (SMTP) engine. To avoid the need to detect and then use disparate email clients on victim systems, worms install their own fully functional SMTP server, ensuring that they can generate infected emails regardless of the email client software used by the victim. This increases a worm's propagation rate.

In contrast to a normal email message generation, a mass-mailing worm automatically composes the infected email message with no human intervention. In fact, unless a virus scanner or some other malicious code detection device detects the infection, the system owner is typically unaware that a worm is resident on the system. Using its built-in SMTP server, the infected system bypasses the corporate mail server when it attempts to send infected emails to the respective recipients.

In this scenario, the SMTP engine of the infected system is responsible for propagating the worm and delivering infected emails. In order to determine the mail server that services a particular recipient, the infected system, not the local mail server, queries the local DNS server for the MX record associated with the email recipient's address. Some worms also contain a list of Internet accessible DNS servers that they can query if communication to the local DNS server from the infected host fails.

Our approach can be used to monitor for MX record queries to uncover systems that query the DNS server directly for MX records. If a local system other than the mail server requests an MX record, we may consider this activity to be anomalous. In order to detect mass-mailing worm propagation, we simply observe all locally generated MX queries to the local DNS server that originate from systems other than the network mail servers. This technique has been validated [74], though we do not claim that published work as part of this thesis.

Anomaly Detection Integration. We have identified that this detection technique can be prone to significant amounts of false positives and negatives when used in an open network environment. In this scenario, we believe this technique could be useful if integrated into more sophisticated anomaly based detectors to avoid false positives and negatives.

6.5 Summary

We present a technique, implemented in a software prototype, to both rapidly and accurately detect worm propagation in an enterprise network. We have demonstrated through our evaluation and analysis that this internal network scanning detection approach can be used in certain network environments to detect L2R scanning activity. Regardless of the scanning rate, the detection algorithm is able to detect a scanning system after only a single scanning attempt. It relies on a network service found in every network (i.e. DNS). The precision of this *first-mile* detection enables the use of automated containment and suppression strategies to stop scanning worms before they leave the network boundary. The DNS-based worm propagation detection approach is an effective way to detect network scanning activity within appropriate enterprise networks. Depending on the network environment and security policy however, the number of protocols added to the whitelist may potentially limit the applicability of this technique as a stand alone detector. In these scenarios, this detection method could be used as an additional detection signal in concert with other worm detection schemes instead of being used as the primary detection technique.

Finally, we believe that this detection approach could be easily modified to detect additional classes of malicious activity including: covert channel detection, mass-mailing worms, automated scanning tools, and remote to local worm propagation.

Chapter 7

Internal Network Scanning Detection Strategy 2: ARP-based

This chapter describes a new technique to detect internal network scanning activity (i.e. L2L) using the address resolution protocol (ARP) [48]. ARP provides a mapping between an IP address to the physical hardware addresses of network devices. We have observed that scanning activity within a network can be detected by observing suspected scanning systems as they generate anomalous ARP requests. Although this technique was designed for the detection of network scanning worms in mind, the scanning methodology employed by known network scanning tools [21, 67, 66] is the same (see Section 7.2.2) and therefore they can be detected using this technique. We describe how this technique can be used to detect internal hosts performing local network scanning as well as present our evaluation results using this technique. This approach has been published in Whyte et al. [72].

7.1 ARP-based Scanning Detection: Approach

Devices that reside within the same network cell use ARP rather than the Domain Name Service (DNS) to communicate. ARP is a layer 2 protocol (i.e. data link layer) used by the IP protocol to map IP addresses to the physical hardware (MAC) addresses of network devices. When a device needs to resolve a given IP address to a MAC address, it broadcasts an ARP request. The ARP request packet contains the sender's IP address (source protocol address), the sender's MAC address (source hardware address), and the destination IP address (target protocol address). Each device within the common broadcast domain receives this packet. The ARP protocol specifies that only the device with the specified destination IP address will respond with an ARP reply. An ARP reply contains both the IP address and MAC address of the device that responds. ARP activity is a necessary precursor to communications between devices as it provides the data link layer with the necessary mappings between the IP and MAC addresses of communicating devices.

An ARP request indicates that a system is trying to resolve an IP address to a MAC address for some type of connection (regardless of the destination port). However, it is possible for a host to connect to a device without an immediately preceding ARP request. Once a device performs an ARP request, the MAC-to-IP address mapping within an ARP reply is maintained locally in the device that receives it in a table called an ARP cache. Only the device that made the ARP request receives the ARP reply. The ARP cache entries also have an associated time to live (TTL) and are dynamically entered and removed. If an ARP reply is received by a device and the MAC address already appears within its cache, it is overwritten by the update. As long as the ARP reply remains in the local cache, subsequent connections to the same device will result in the MAC address being obtained from the cache rather than through an ARP request. The affect of ARP caches on our approach is discussed in Section 7.2.3. Our technique can be deployed on any device within the broadcast domain, even in a switched network fabric, and as it only processes ARP requests, it is extremely efficient.

L2L worm propagation can occur within a particular cell or span multiple cells depending on the scanning strategy of the worm. L2L scanning activity results in unusual ARP activity, namely: (1) an infected device will use ARP to try to connect to some devices within the internal network with which it had no previous history of connecting to pre-infection; (2) the number of ARP requests generated per fixed unit time (e.g. every 60 seconds) will increase; and (3) in those networks where not all IP addresses within a netblock have been allocated, ARP requests will be generated for nonexistent systems (i.e. for *internal network dark space*). We now discuss in turn how we use each of these behaviors to derive an aggregate anomaly score for each device within a cell.

Peer List (Customary ARP request targets). A training period is used to determine normal device interactions within a cell. ARP requests observed during the training period allow us to characterize normal device interactions through the use of a peer list (see Figure 7.1). A peer list is indexed by all the devices which made ARP requests (i.e. served as a source protocol address within an ARP request) during a training period. Each ARP chain contains entries of the devices being queried in requests by the indexing device (i.e. target protocol address within the ARP request).

The source protocol address of an ARP request is the IP address of the device trying to resolve an IP address to its corresponding MAC address. The target protocol address, encapsulated within an ARP request, is the IP address of the system being queried which will be the eventual target of a connection. Each time an ARP request is generated, any observed new source protocol address is recorded as an index entry within the peer list. Figure 7.1 contains four entries within its index (i.e. 192.168.1.1, 192.168.1.2, 192.168.1.5, and 192.168.1.9). The four ARP chains contain 2, 3, 1 and 3 elements respectively. Each ARP chain is comprised of an index value equal to an ARP requester's IP address. The index and the associated collection of ARP chains is known collectively as the peer list. The corresponding target protocol addresses of the respective queries are added as entries indexed by the corresponding source protocol address. Over a training period, we build an index of active systems within the network cell (i.e. ARP requesters) and the list of devices (*ARP chains*) they are trying to connect with. The individual elements within each ARP chain are derived from the set of IP addresses queried by the ARP requester. If an ARP requester queries the same device more than once, this activity is ignored (i.e. no duplicate entries exist within an ARP chain). Typically, individual devices will only communicate with a small subset of other internal devices that offer some sort of service (e.g. DNS, file, router, etc.).[7]

For each device i in a cell, we assign an anomaly score for the *peer list factor* (a_1) as: $a_{1,i} = x$ where x is the number of ARP requests as made by device i (in the current sample interval) which are outside of device i's ARP chain. In our testbed, the device identifier i corresponds to the last octet in the device's IP address. For instance, if two ARP requests are made by device 192.168.1.9 to two different IP addresses not contained within its ARP chain within the current sample interval, then $a_{1,9} = 2$. Subsequent distinct connection attempts outside a device's ARP chain within the detection window (see Setting Alert Thresholds in this section) result in a linear growth for this anomaly factor.

[7]Of course, this assumption is violated e.g. in P2P networks and highly distributed cooperative network environments. We discuss this further in Section 7.3.

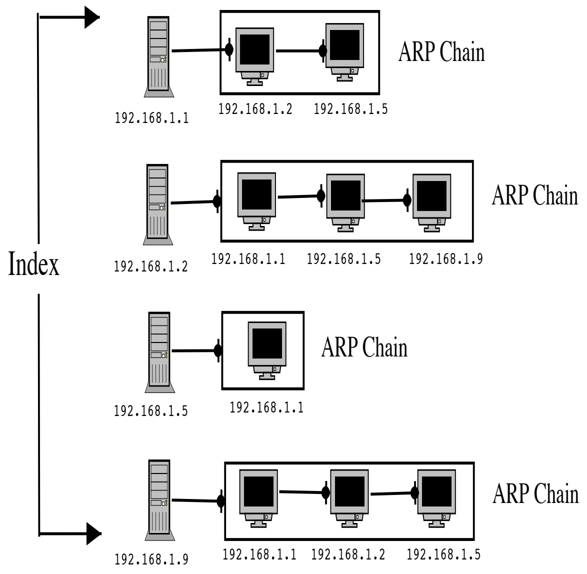


Figure 7.1: Peer List For a Small Network With Four Active Devices.

The running total of the a_1 factor for device i is the sum of the $a_{1,i}$ values over all sample intervals in the current detection window. More specifically, at sample interval j, let $a_{1,i}$ as defined above be shorthand for $a_{1,i}^{(j)}$; then the running total for $a_{1,i}$ for the current detection window of width w is $A_{1,i} = \sum_{k=0}^{w-1} a_{1,i}^{(j-k)}$.

If a device tries to connect to another device outside of both its ARP chain and the peer list, it is assigned an anomaly score based on the internal network dark space factor described in this section. Note that a possible source of false positives results from the fact that although perhaps suspicious, a device may legitimately contact another device outside of its ARP chain (e.g. a legitimate infrequently occurring interaction between two devices may not be captured during the training period thus not all such occurrences are necessarily an indication of a scanning worm).

ARP Activity (Number of ARP requests). The number of ARP requests is recorded for each active device within the network over discrete sample intervals (e.g. 60 seconds) during the training period. Once the training period is complete, the mean ($\overline{x}$) and standard deviation (σ) of the observed ARP request activity are calculated for each individual device. The standard deviation reveals the normal fluctuation in ARP activity that can be expected during sample intervals.

We set a (somewhat arbitrary) upper bound and call it the *expected maximum*

ARP request activity (E_i) *for device* i within a sample interval:

$$E_i = \overline{x} + 2\sigma \tag{7.1}$$

A primary factor in choosing this value is that in a normal (i.e. Gaussian) distribution, 95 % of the data values will fall within two standard deviations of the mean value; however, it should be clear that other selections may be equally or more useful. Once the training period has ended, the observed (i.e. subsequently monitored) ARP request activity O_i for each device i, is compared to E_i. $O_i > E_i$ may indicate anomalous scanning activity.

We assign an anomaly score for the *ARP activity factor* (a_2) for device i as follows:

$$a_{2,i} = \begin{cases} O_i - E_i & ;\text{if } \mathrm{O_i > E_i} \\ 0 & ;\text{if } \mathrm{O_i \leq E_i} \end{cases} \tag{7.2}$$

Similar to the a_1 factor, this calculation is performed during each sample interval to determine a *running* total within the *detection window* (see Setting Alert Thresholds in this section) for each device. As device i's ARP request activity (O_i) increases beyond E_i, $a_{2,i}$ increases linearly. For instance, server 192.168.1.11 has $\overline{x} = 1.57974$, $\sigma = 1.03620$, and $E_{11} = 3.65214$. If $O_{11} = 5$ then $a_{2,11} = 1.34186$. This factor is particularly useful in addressing the affect of local ARP caches and large ARP chains (as discussed in Section 7.2.3).

Internal Network Dark Space. Internal network dark space is defined during the training period. Looking at the peer list in its entirety, we derive a set of internal system addresses that comprise the active systems within the cell during the training period. ARP requests for IP addresses not contained within this set we consider to be anomalous, and refer to as *internal network dark space.*

Each network consists of a block of network addresses. Although it is not unusual for an internal network to consist of IPv4 Internet addresses, most internal networks use non-routable IP address schemes. If RFC 1918 [53] is followed, the internal address space can be as large as a Class A network. This allows internal networks to consist of potentially millions of hosts. Regardless of whether the IP addresses within a network are non-routable or Internet-accessible, there may exist IP addresses that are not bound to any device within the cell. Connection attempts to these vacant IP addresses are anomalous. We assign an anomaly score for the *internal network dark space factor* (a_3) during a given sample interval as:

$$a_{3,i} = \begin{cases} 0 & \text{; if no dark space scans} \\ y & \text{; if device i scans dark space} \end{cases} \tag{7.3}$$

We suggest that the value y be assigned such that it is the same for all devices (i) and a single observed connection to an internal network dark space address should generate a value $a_{3,i}$ sufficient on its own to meet the *alert threshold* r and generate an alarm (i.e. in our prototype, we set $y = r$; see Setting Alert Thresholds below).

SETTING ALERT THRESHOLDS. Our implementation requires that a scanning worm exhibit a minimum sustained scanning threshold of one scan per minute. Therefore, we define a sample interval as 60 seconds (i.e. $t = 60$ seconds). The choice of sample interval directly affects the amount of state information that must be maintained by the prototype. The *detection window of width* w (= number of sample intervals) is the period of time in which observed anomaly scores for factors a_1 and a_2 must be maintained in state. In our implementation, we set $w = r$ (see definition of r below).

For example, for $r = 1$ the detection window is 60 seconds and an alert is generated upon a single anomalous scan observed within a one minute period. If we set $r = 2$, the detection window is 120 seconds and two anomalous scans must occur within two minutes to trigger an alarm. Anomalous scans get aged out over time; scans which slide out as the current detection window moves no longer contribute to the anomaly score. Anomaly scores are attributed to devices as ARP requests are processed by the prototype. Therefore, alarms can be generated at any time regardless of the size of the detection window or when they are observed within a sample interval.

For each sample interval and each device, we derive the total anomaly score for device i as:

$$a_{T,i} = a_{1,i} + a_{2,i} + a_{3,i} \tag{7.4}$$

If the current sample interval is denoted as sample interval j, and $a_{T,i}$ above is shorthand for $a_{T,i}^{(j)}$, then the total anomaly score for a window of width w ending at sample interval j is:

$$A_i = \sum_{k=0}^{w-1} a_{T,i}^{(j-k)} \tag{7.5}$$

An alarm is generated when $A_i \geq r$ for any device i. With respect to factor a_1 and a_2 (or a combination thereof), the configurable alert threshold (r) for A_i is the minimum number of anomalous scans that must be made by device i within the detection window before an alarm is generated. As usual, a balance must be struck between incurring false positives and negatives (refer to Section 7.2.3). The lower the alert threshold, the more sensitive to ARP fluctuations the detection system becomes and the greater the possibility of a false positive. r can be manually set before the training period or automatically determined by the detection system. Our prototype automatically sets r to the floor of the highest E_i value it has calculated over the training period. For instance (cf. Table 7.1), 192.168.1.11 has the highest E_i score (3.65214) therefore $r = 3$. We discuss how the three anomaly factors interact next to produce A_i.

The first two factors (a_1, a_2) reflect the number of anomalous scans observed within the detection window. An anomalous scan for device i is considered to be any of: (1) each ARP request outside of a device's ARP chain (a_1 factor); (2) each scan by which O_i is in excess of E_i (a_2 factor); or (3) the detection of a single scan to internal network dark space (a_3 factor). The third factor is unique in that per our parameter settings, an alarm was triggered after a single observation of a scan to internal network dark space regardless of the value of r and the time of observation. For instance, if $r = 2$ an alert will be generated after detection of two anomalous scans within the detection window (i.e. $A_{1,i} + A_{2,i} = r$) or the single observation of a scan to internal network dark space (i.e. $A_{3,i} = y = r$).

7.1.1 High-Level System Design

Our software implementation uses the *libpcap* [2] library and is comprised of two logical components: PPE and ACE. The Packet Processing Engine (PPE) is responsible for extracting the relevant features from the live network activity or saved network trace files. The ARP Correlation Engine (ACE) includes a dynamically generated peer list and the list of IP addresses it considers to be internal network dark space.

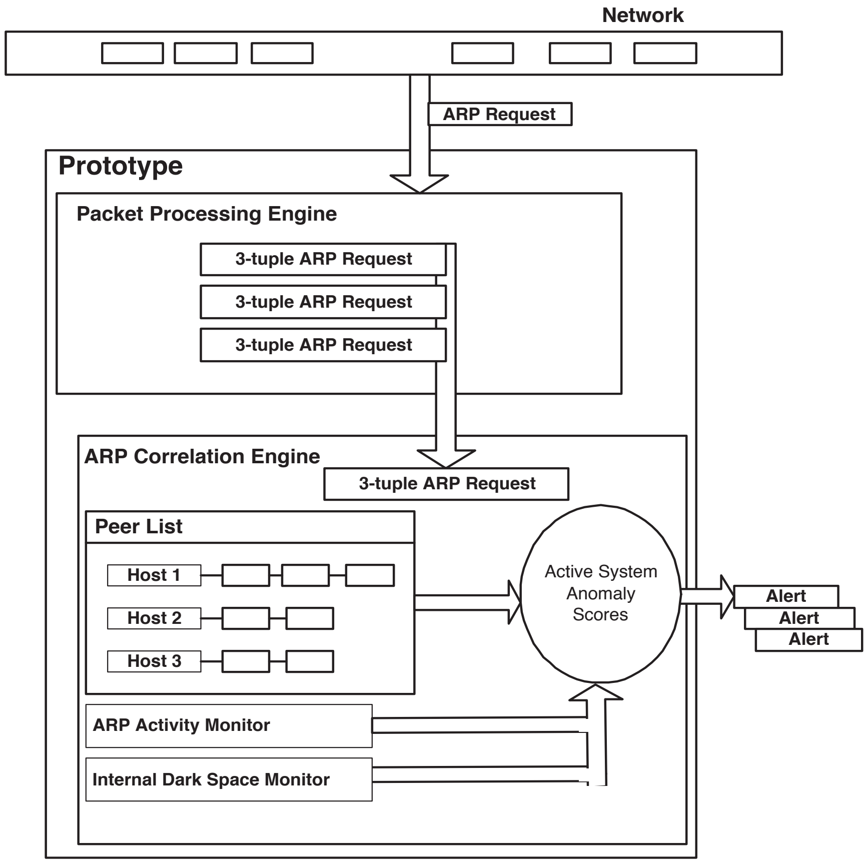


Figure 7.2: High Level Design of Prototype Implementation

The ACE maintains in state all relevant ARP information extracted by the PPE. To detect worm propagation between network cells, an instance of such a prototype would need to be deployed in each broadcast domain. Figure 7.2 shows the high-level design.

Packet Processing Engine (PPE). The PPE is responsible for extracting all ARP request packets from network capture files or live off the network. Due to the transmission mechanism of ARP requests (i.e. broadcast), the prototype can be deployed on any device within the network cell. Other forms of ARP activity (e.g. ARP replies) are ignored making this scheme stateless. Feature extraction from the ARP request packets includes 3-tuple tokens (source IP address, target IP address, timestamp) which are passed to the ACE for processing.

ARP Correlation Engine (ACE). The ACE processes all network features passed to it by the PPE. The ACE is responsible for four major functions. During the training period, the ACE: (1) creates individual-device specific ARP request statistics, and (2) creates the peer list. Once the training period is complete the ACE: (3) uses ARP request activity to generate a three-factor anomaly score, and (4) generates alarms when the alert threshold has been met or exceeded.

Network ARP Statistic Extraction. During the training period, all tokens passed by the PPE are processed to determine system specific ARP request activity. The ACE maintains ARP request statistics for each active device (i) within the network. ARP requests, encapsulated in tokens from the PPE, are processed in sampling intervals of duration t. For our implementation, we chose a value of t such that it matched the default ARP cache time to live (ttl) of our devices (i.e. Linux operating systems). Therefore the mean and standard deviation for number of ARP requests is made per sample interval of 60 seconds during the training period. Max ARP requests in Table 7.1 refers to the maximum observed ARP requests over any sample interval during the training period. If within a sampling interval there is no ARP request activity, this observation is excluded from final mean and standard deviation calculations. This is to compensate for frequent periods of inactivity (e.g. nights and weekends) that would skew the ARP request statistics giving them lower values than in peak usage times. At the end of the training period, the mean and standard deviation of ARP request activity is calculated for each device (see for example Table 7.1). These values comprise the expected maximum (E_i) ARP request activity for each individual device within the cell.

Peer List. The peer list, constructed during the training period, contains a listing of all *live* devices and the IP addresses of the internal devices they were in communication with. In a typical network environment (i.e. client server model), internal devices will try to access only a subset of the devices within their local subnet, as recorded by their respective ARP chains. For any given device, connecting to a device within its respective ARP chain should occur at a higher probability than other devices within the peer list. If a worm performs unrestricted scanning of a network cell, anomalous ARP activity will result. First, ARP requests between system pairs (i.e. infected host and a subset of victims) that have not been observed communicating during the training period will occur. Second, as an infected host tries to connect to internal devices the number of ARP requests it issues per unit time will increase.

Anomaly Score and Generating Alerts. Once the training period is complete, an anomaly score for each individual device within the network cell is maintained (see Section 7.1). An alert is generated when $A_i \geq r$ for any device i. The timestamp from the triggering ARP request is used as the timestamp for the alert, which also indicates the alert triggering source and destination address (see Figure 7.3).

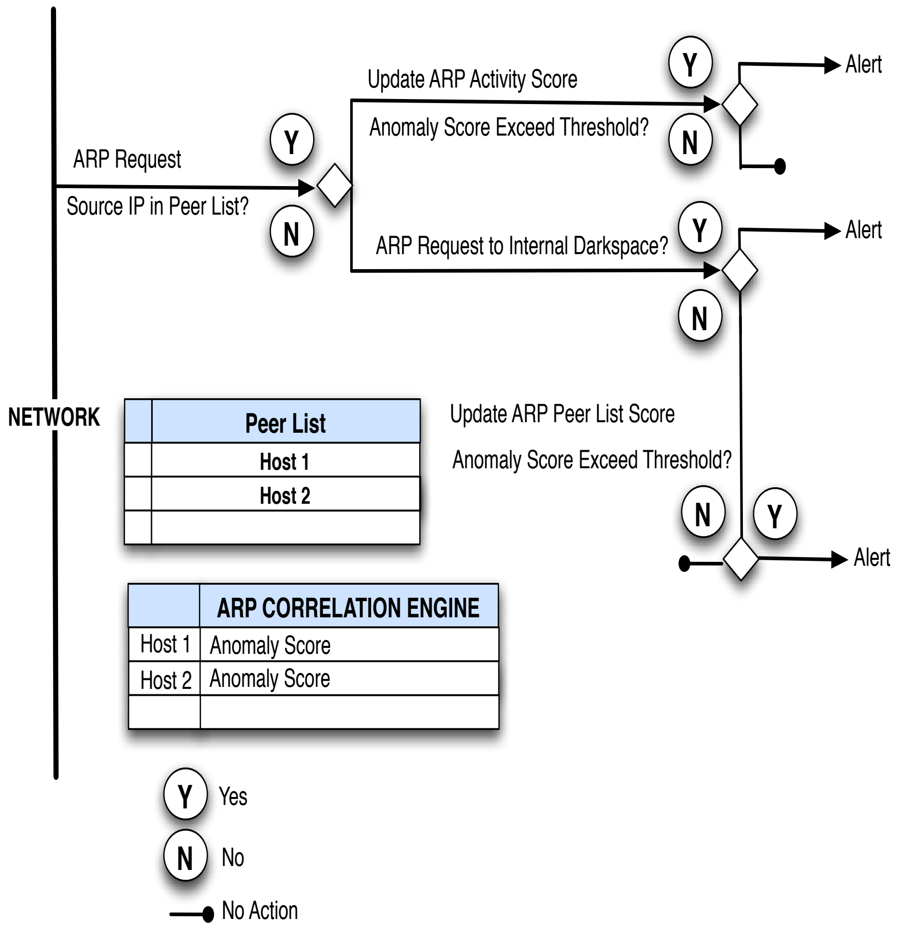


Figure 7.3: ARP-based Detection Logic.

Table 7.1: ARP Statistics for Prototype System on CCSL Network.

Servers					
IP Address	ARP Requests	ARP Chain Size	Mean	Standard Deviation	Max ARP Requests
192.168.1.11	13 532	21	1.57974	1.03620	8
192.168.1.12	13 335	17	1.20305	0.61021	9
192.168.1.13	11 318	16	1.17641	0.42872	4
192.168.1.14	2 701	2	1.01310	0.11376	2
Workstations					
IP Address	ARP Requests	ARP Chain Size	Mean	Standard Deviation	Max ARP Requests
192.168.1.15	10 216	8	1.29761	0.51830	4
192.168.1.16	8 254	10	1.24309	0.46756	4
192.168.1.20	9 080	9	1.21419	0.46745	5
192.168.1.21	3 335	8	1.17489	0.47421	5
192.168.1.22	6 016	9	1.21649	0.44508	3
192.168.1.23	5 513	10	1.20426	0.44432	4
192.168.1.24	6 863	8	1.17125	0.42354	4
192.168.1.25	5 546	9	1.20074	0.43677	5
192.168.1.26	5 642	11	1.06825	0.26134	3
192.168.1.27	2 311	9	1.19744	0.46979	5
192.168.1.30	7 708	5	1.72427	0.67429	4
192.168.1.31	5 746	4	1.52241	0.63850	3
192.168.1.33	17 782	6	1.23520	0.44930	3
192.168.1.45	57	4	1.07547	0.26667	2
192.168.1.51	99	4	1.42307	0.75006	4
192.168.1.52	10	5	1.45454	0.93419	4

7.2 Prototype Evaluation

In this section, we describe the network and data set (network traffic) we used with our software prototype as a proof-of-concept to validate our proposal and refine our system design, and discuss how our prototype performed. We did not perform a side-by-side comparison of this technique with another L2L scanning detection technique as, to the best of our knowledge, no comparable publicly available L2L scanning detection technique exists. Four weeks of network traffic was collected in one of our university research labs (i.e. the Carleton Computer Security Lab – (CCSL)). The first two weeks of the network data set was used as the training period. We then tested the prototype on the remaining two weeks of data, to determine both the validity of our detection technique, and the affect of the configured alert thresholds on false positive rates. We tested two different approaches to setting alert thresholds:

i. Common threshold approach (Section 7.2.1): give every device within the network cell the same alert threshold r.

ii. Function-specific threshold approach (Section 7.2.1): partition the network cell to give devices that perform different functions (e.g. server, workstation) different alert thresholds r_j where j is the function used to partition the network cell.

Finally, we will describe our scanning worm simulations and report on the performance of our detection software in detecting these scans.

7.2.1 Data Set for Prototype Evaluation

To validate our approach, we developed and tested a fully functional software prototype with all features discussed in Section 7.1. The software was installed on a commodity PC running Linux with a 10/100 network interface card. The CCSL network consisted of a one quarter Class C network of Internet-reachable IPv4 addresses (i.e. 62). Using the cell definition from Section 7.1, the CCSL network contained one cell. The CCSL network analysis allowed us to test the prototype's ability to detect L2L intra-cell worm propagation. Network traffic was collected from November 11 to December 11, 2004.

From the two week training period the prototype automatically determined each device's peer list size, mean number of ARP requests per minute, standard deviation of ARP requests per minute, and the largest number of ARP requests observed by each device within the sampling interval (we used a 60 second period). The last characteristic is not used in the determination of the anomaly score but as an input to analyze the effectiveness of the approach as previously discussed. After the training period we recorded, for analysis, ARP activity within the network cell in a single *pcap* file for the next two weeks. During this analysis period, we monitored the CCSL network independently with an intrusion detection system (i.e. snort [56]) to ensure no worm activity was included within the data set. Finally, we simulated scanning worm propagation within the CCSL network using the *Nmap* [21] security scanner to test our detection software.

The respective ARP request activity for each active system within the CCSL network is included in Table 7.1. Network infrastructure devices (i.e. firewall and switches) were excluded from analysis and thus do not appear as index entries within the peer list. The total size of ARP request traffic for the entire four week period was approximately 9 MB.

Approach 1: Common Threshold. Table 7.1 separates servers and workstations in our testbed. Note that, with the exception of one server (a secure log server), the servers within the network have the largest ARP chains. The device with the largest peer list (i.e. 192.168.1.11) was the DNS/mail server for the CCSL network. This is not unexpected in a typical client-server model. Likewise, the servers within the CCSL network also had the largest observed ARP requests within the sampling intervals.

By applying our technique on all the devices within the network, we determined the number of false alarms generated as a function of our alert threshold. Applying a common alert threshold to all devices (*common threshold approach*), we ran the prototype on the second two weeks of archived ARP request data, varying this threshold to observe the affect on false positive rates. Each trial *run* of the prototype (i.e. processing a two week data file) took less than one minute to complete. A subset of our results are captured in Table 7.2.

Setting $r = 1$ resulted in 99 false positives over the two week dataset. Recall that a scan to an IP address considered to be internal network dark space was set to immediately generate an alarm regardless of r by our suggested configuration of $y = r$. As expected, we observed no scans from internal devices to internal network dark spaces.

Setting the alert threshold at $r = 2$ causes an alarm to be generated after observing two anomalous scans within two time intervals (i.e. 2 minutes). With $r = 2$, 22 false positives resulted over the two-week period. For contrast, for $r = 3$ (the value automatically selected by the prototype, see Setting Alert Thresholds), only 5 false positives resulted over the two-week period. For use in an automated response system, the occurrence of 5 false positives within a two-week period may be too great. In our test network, as we increased r, the number of false positives decreased. If we manually set $r = 6$ only 1 false positive from all devices is generated within a two-week period.

Approach 2: Function-Specific Thresholds. One method to refine our approach is to use different alert thresholds, for different categories of devices based on the system function (not currently implemented by our prototype). For instance, we observed that most servers (with one exception) have a higher ARP chain counts than workstations. Additionally, two servers have the two highest observed per minute ARP request counts during the training period (servers 192.168.1.11 and 192.168.1.12, see Table 7.1). Not surprisingly, a server must be able to handle bursts of requests from other network devices. However, legitimate bursts in ARP request activity may cause the a_2 factor to exceed its alert threshold causing false positives. All 5 false positives generated at the $r = 3$ threshold were caused by the two servers with the highest per-minute ARP request count. We refer to distinguishing of devices within the network cell to allow differing alert thresholds based on the function of a device (e.g. server or workstation) as the *function-specific thresholds* approach. We could allow the workstations alarm threshold to be set at $r = 3$ reducing their false positive rate to zero (i.e. for our test network). Increasing the server alarm threshold to $r = 5$ reduced not only their false positive rate but the overall false positive rate for the two-week period to 2.

Varying alarm thresholds could be extended to other classes of systems within the network cell if required, depending on the nature of applications running on the network. An example would be relaxing r on devices that use distributed or P2P applications. We expect that these devices would repeatedly initiate a greater number of connections (ARP requests) than other devices during the sampling intervals. This increased activity would typically give these devices larger mean and standard deviation scores causing higher E_i values. Devices with larger E_i values will require the observation of more ARP requests than less active devices (i.e. lower E_i) before r will be met or exceeded. In this scenario, the ability to assign differing r values to

Table 7.2: Alarm Threshold Analysis

Alerts			
Threshold r	Server	Workstation	Total
1 scan	37	62	99
2 scans	19	3	22
3 scans	5	0	5
4 scans	3	0	3
5 scans	2	0	2
6 scans	1	0	1

Anomalous Connection Activity			
	Server	Workstation	Total
Outside ARP chain	181	36	217
Dark space	0	0	0

devices with similar E_i would minimize both false positives and negatives. However, as r increases so does the number of worm scans before an alarm is generated.

7.2.2 Simulating Scanning Worm Activity

To simulate scanning worm propagation within a network cell we used the port scan option of the Nmap security scanner. Just like a worm, the kernel and networking components of the workstation performing Nmap scans use ARP in order to make contact with the devices within the network cell. We configured Nmap to scan a single port (port 80/TCP) on all the devices within the network cell. Ignoring the two broadcast addresses left 62 usable IP addresses. These port scans simulated a scanning worm trying to find versions of vulnerable HTTP servers within the network cell. The device we used to scan the network cell was a workstation that had the highest ARP chain count (192.168.1.26 had 11 ARP chain entries; see Table 7.1). We set our alert threshold to $r = 3$. Based on Section 7.2.1, this was the minimum threshold that incurred no false positives from workstations during testing and it was also the value automatically selected by our prototype. To fully exercise our detection software, Nmap was run in two modes, each with two types of scanning strategies as follows.

The first two tests consisted of scanning port 80 on every device within the network cell using Nmap's *normal mode* (scanning the network as quickly as possible) employing both the sequential and random scanning strategies. The last two tests consisted of scanning port 80 on every system within the network cell using *sneaky mode* (waiting 15 seconds between scans in an effort to become stealthy) employing both the sequential and random scanning strategies.

Table 7.3: Network ARP Statistics

Number of Scans Before Detection		
	Normal	Sneaky
Sequential	2	2
Random	1	3

Worm Simulation Results. Nmap Sequential Scanning Strategy. The sequential scans for both normal and sneaky mode were detected within two port scans (see Table 7.3). Nmap was configured to sequentially scan the host range from 192.168.1.1 until 192.168.1.62 (thus omitting network broadcast addresses). 192.168.1.2 was the target of the second scan in both sequential scans. IP address 192.168.1.2 was assigned to a network switch and does not appear within the peer list and therefore is considered internal network dark space. In these cases, sequential scanning detection was triggered by the $a_{3,26}$ factor within the aggregate anomaly score for the system.

Nmap Random Scanning Strategy. In normal mode, the random scan was detected within one scan. Of the 62 usable addresses within the network, the total peer list size of servers and workstations was only 21 (i.e. approximately 66% of our network was defined during the training period to be internal network dark space). The first random scan in normal mode was to an internal network dark space. Again, the a_3 factor dominated the aggregate anomaly score and caused an alarm to be generated after the first scan.

In sneaky mode, the random scan was detected within three scans. In this case, although statistically improbable, no internal dark space addresses were scanned. The $a_{2,26}$ factor became the dominant factor and triggered after detecting three ARP requests above E_i for the device within a three minute period. However, it is statistically probable that subsequent tests using the same parameters would be detected by the a_3 factor before three scans.

The Affect of Dark Space on Sequential and Random Scanning Detection. Overall, our detection testbed implementation benefited from the sparse internal IP addressing scheme within the network cell. Internal network dark space comprised approximately 66% of the network cell's usable IP addresses. If p is the probability that a random scan will be to internal network dark space then $1 - p$ is the probability that a random scan will not be to internal network dark space (e.g. 0.3387). Random scans are independent events. The probability that the $a_{3,i}$ factor will trigger causing an alarm after the occurrence of three random scans is $1 - (1-p)^r = 1 - .3387^3 = 0.9611$.

The large amount of internal network dark space also aided our prototype in detecting sequential scanning. Topological worms typically harvest network configuration information from their victims for new targets [69]. In our testbed, any sequential scanning strategy that started from the lowest IP value within a device's network subnet configuration value (i.e. 192.168.1.1) would be detected within the second scan.

7.2.3 Discussion of False Positives and Negatives

We now discuss the impact and causes of false positives and negatives on our detection technique. Since the analysis is valid for both the common threshold and function specific thresholds approaches, we discuss only the specific results of the approach with the greatest number of false positives (i.e. common threshold approach).

False Positives. All five false positives which arose when the alert threshold was set to $r = 3$ (triggering at r or more scans) were caused by servers, and specifically by bursts in server activity. A typical scenario for normal network activity involves users logging onto their workstations and requesting network services (e.g. DNS, mail, etc.) that allow them to execute desired tasks. As a workstation generates an ARP request to determine the MAC address of the server, the server also typically generates ARP requests to determine the MAC address of other servers that assist them in performing their tasks. When a number of users access services simultaneously, this will cause a burst in ARP requests from the servers. The two most active servers (i.e. 192.168.1.11 and 192.168.1.12) in our testbed have the highest observed maximum ARP requests in the sampling interval. ARP request bursts caused by servers answering legitimate service requests can produce false positives. The occurrence of false positives could be reduced by raising r. However, each increment of r allows another scanning worm infection attempt to occur before an alarm is raised.

Automated attack and scanning tools share the same searching strategies as scanning worms. These tools can perform random or sequential scanning at differing speeds to either exploit or identify vulnerable systems. Our ARP-based detection technique discovers intra-cell scans caused by such tools (e.g. if the minimum sustained scanning rate exceeds one scan per minute), but does not distinguish them from scans resulting from a scanning worm.

False Negatives. A false negative occurs when malicious activity occurs without triggering an alarm. In this section, we discuss the affect of ARP caches and large ARP chains on possible false negatives for our detection technique.

ARP Cache. If a device happens to have the MAC address of the device it wants to communicate with within its local cache, no ARP request is generated. For this reason, typical scanning worms exploit an infected device's local ARP cache just as any legitimate network application would. However, our prototype is network-based and does not have access to the local ARP caches of the devices within the network cell. For this reason, with respect to the $a_{2,i}$ factor, this activity is not reflected within the O_i activity count.

However, our calculation of E_i (see equation (7.1), Section 7.1) does offer some insight on the affect of ARP caches. For example, system 192.168.1.30 (statistically the most active workstation in Table 7.1) has $\overline{x} = 1.72427$ and $\sigma = 0.67429$, making $E_{30} = 3.07285$. This represents the maximum *expected* number of ARP requests a device can make within a one minute period without causing a positive $a_{1,30}$ score. Recall (see Section 7.1.1) that the default ttl for the entries within the ARP caches of the devices in our network cell is 60 seconds, which matches our $a_{2,i}$ sampling interval. ARP replies (i.e. MAC and IP address pairs) are cached locally on the devices that generated the associated ARP requests.

According to E_i, we expect that three is a reasonable upper bound for the number of entries within this device's ARP cache. If a scanning worm happened to select any of the IP addresses within the cache, no network ARP request would be sent and our prototype would not detect the scan (i.e. O_i would not be incremented by 1). ARP caches can be a source of false negatives. However, in our calculations of $\overline{x}$ and σ for each device, we ignored the affect of long periods of inactivity. This was done to better approximate the ARP request activity during active usage (i.e. ignoring user and system inactivity) to ensure that our false positive rates would be minimized for this factor. E_i represents a reasonable upper bound to the number of entries within the respective local ARP caches. In practice, the ARP caches will typically contain fewer entries than the E_i values for each system since by design, E_i exceeds the corresponding mean value. Therefore, we expect that ARP caches have a minimal affect on factor $a_{2,1}$.

Additionally, the $a_{2,i}$ factor is not applied in isolation. In order to avoid any anomaly score contribution from the remaining two anomaly score factors (i.e. $a_{1,i}$ and $a_{3,i}$) all these scans would have to be limited to the device's ARP chain. The probability of detection avoidance through limiting scans to the respective ARP chain is discussed in the next section.

Large ARP Chain. We use ARP chains to characterize normal network interactions. A system with a large ARP chain will be able to connect to the devices within the chain without contributing to the $a_{1,i}$ factor. The largest ARP chain in our testbed belonged to the server 192.168.1.11 with 21 entries. The largest workstation ARP chain belongs to 192.168.1.26 with 11 entries. Table 7.4 shows the probabilities of the two systems scanning within their respective ARP chains, internal network dark space, and their peer lists.

As previously discussed, internal network dark space dominates the overall scanning possibilities for worms within our testbed network cell. Even the device with the largest ARP chain (192.168.1.11) had only a 0.3387 probability of selecting an IP address within its ARP chain. During a random scan, the chance of 3 successive scans all targeting IP addresses in the ARP chain is approximately $(1-p)^3 = 0.3387^3 = 0.03885$ (see Section 7.2.1). If a sequential scanning strategy was used, the probability that 3 successive scans would all be to devices within an ARP chain would depend on the IP address composition of the ARP chain. Using the CCSL network as a practical

Table 7.4: Anomaly Factor Triggering Probabilities in Testbed

System Specific ARP Request Statistics			
	ARP Chain Size	$\overline{x}$	σ
192.168.1.11	11	1.57974	1.03620
192.168.1.26	21	1.06825	0.26134
Scan Location Probability			
	ARP Chain	Dark Space	Peer List
192.168.1.11	0.3387	0.6613	0
192.168.1.26	0.1774	0.6613	0.1613

example, a sequential scan from 192.168.1.11 would deviate from its ARP chain on the second scan (see Section 7.2.2) and be detected by $a_{1,11}$ and $a_{3,11}$. Regardless, if a scanning worm managed to only scan IP addresses within the device's ARP chain, it would have to do so with a sustained scanning rate of less than 1 scan per minute or it would be detected by $a_{2,11}$. Therefore, the application of a_2 ensures large ARP chains have a minimal affect on the detection technique.

7.3 Limitations

LIMITATIONS. Our detection technique cannot detect L2L inter-cell, R2L and L2R worm propagation. It would be useful as a complementary technique used in conjunction with other approaches, e.g. [71] to detect L2L intra-cell scanning worm propagation. Furthermore, our approach relies solely on the observation of ARP requests. We do not try to match the associated ARP replies to determine if the subject of the ARP requests are actually active on the network. In the event that a system broadcasts an ARP request to a device currently not active on the network (e.g. powered down or disconnected from the network) due to scheduled maintenance or some unscheduled failure, this will not be considered a scan to internal network dark space as long as its address was observed during the training period. In this scenario, the amount of internal network dark space would be understated as inactive devices would be considered *live*. If we extended our approach to correlate ARP requests and replies we could determine which devices are live or actually internal network dark space.

ARP request and reply correlation would also address another potential limitation that arises in networks that use the Dynamic Host Configuration Protocol (DHCP) [17]. DHCP allows network devices to determine their IP addresses from a central server rather than from a static configuration file. When a device becomes active on the network, it contacts its DHCP server to retrieve an IP address that it can use on the network. The MAC address of the requesting device is then associated with an IP address assigned by the DHCP server. DHCP-assigned IP addresses are *leased* to the devices that request them. A DHCP lease is the amount of time that the DHCP server grants permission for a device to use a particular IP address. The devices in our test network used static IP addresses and thus the MAC and IP pairing was constant. In a DHCP-enabled network, the MAC address and IP pairing is not guaranteed to be constant (e.g. when the lease expires a device may receive a different IP address and thus the IP address MAC pairing is different). Our prototype uses IP addresses to identify devices and thus would be adversely affected by allowing a device to have different IP addresses. ARP request and reply correlation would enable us to use the MAC addresses (which are fixed and never change) of devices for identification which DHCP has no affect on.

Another limitation is that network dark space addresses are determined by observing ARP requests on the network and building a peer list. Once the training period is completed, in our description thus far there is no mechanism to add to the peer list or determine if previously active devices have been taken off the network. This may provide an inaccurate accounting of internal network dark space. To address this limitation, we could correlate ARP requests and replies to determine the emergence of new devices. Currently, if we observe an ARP request to an IP address outside the peer list an alarm is generated. If we modified our approach to consider ARP replies, we could dynamically determine internal network dark space.

In a P2P or distributed computing environment, network devices may interact with a large number of other devices. The ARP chains for the devices could be quite large and homogeneous. In this scenario, the a_1 factor would be affected as a device could interact with a large percentage of the network cell and still remain within its ARP chain. Furthermore, if the device was involved in performing tasks that required frequent interaction with multiple devices over long periods of time its $\overline{x}$ and σ would be large. At a minimum, this would require our prototype to observe a greater number of worm scans before the a_2 factor would trigger. However, in a network environment where frequent and varied P2P activity is prevalent, this technique may be limited in its ability to discern *normal* P2P activity from scanning activity.

Attempted Circumvention. A possible worm infection strategy would be to only perform infection attempts after the device had initiated a connection through legitimate use. In this scenario, a scanning attempt could be initiated when the IP address is in the local cache thus obviating the need for an ARP request. A slight modification to this strategy would be for the worm to install itself on a host and monitor ARP request activity before propagation. Worm propagation could then be restricted to those devices that were the subject of ARP requests. In these scenarios, the a_1 and a_3 factors would not be affected by this activity. To ensure stealth and evade detection by this technique, once the propagation began, if the worm's sustained scanning rate was kept to less than 1 scan per minute, factor a_2 would be unaffected as well.

7.4 Suppression and Containment Extensions

To suppress and contain scanning worms within a network cell apparently requires that one employ an automated active response. Two scanning worm containment strategies are: (1) stop the scans as they traverse the network before they reach their intended victims; and (2) stop the scans before they leave the infected host. The first strategy could be achieved through integrating a containment capability directly into the network fabric (e.g. switches). The second strategy could be achieved through integrating a containment capability on each host itself. We believe that our ARP-based detection technique could be used in either strategy to enable automatic containment. Each strategy has its benefits and limitations; we discuss these in turn.

Integration into Network Fabric. A network switch provides a channel for incoming data from any one of its input ports to a specific output port that connects a device to the network. Most network switches have the capability to maintain statistics on the type and amount of network data passing through its ports.

We believe our ARP-based technique could be integrated into switches to allow for monitoring of ARP activity on all the devices they connect to the network. Switch integration would enable us to address a current limitation of our approach by allowing ARP requests and replies to be correlated (see Section 7.3). ARP replies are not broadcast and must be observed at each individual device at which they are received or on the switches that forward them. Integration of our detection technique into the switching fabric would enable us to perform a more complex correlation for better detection and better provide the ability to take active response. For instance, if worm activity was detected on a specific port, a switch could simply turn the port off isolating the device connected to that port from the rest of the network.

A network-based detection approach such as this does have some advantages over a host-based approach. A network-based device can be hardened against attack and is typically administered by security conscious personnel. In a host-based detection approach, the host is the subject of the attack and there is a risk that a successful attack may include turning detection software off or instructing it to not alarm. Furthermore, administration and reporting of alarms is typically easier in network-based detection approaches as fewer devices need to be maintained and monitored.

Host-based Worm Containment. The success or failure of a worm infection is more easily determined at the host than at the network layer. In a network-based approach, one may see an attack go by without knowing if it succeeded, unless one observes subsequent anomalous activity (i.e. attacks) from the suspected victim. Our ARP-based technique could be integrated into a host's networking software or hardware to enable containment. As in Section 7.1, a host-based integration would also enable us to address a current limitation of our approach by allowing us to correlate ARP requests and ARP replies (see Section 7.3). ARP replies received at the individual host would enable us to perform a more complex ARP correlation for better detection accuracy. Again, as in Section 7.1, a host-based approach would also give us the capability of an active response on the intended victim. For instance, if worm activity is detected on the individual host, the host could remove itself from the network by not forwarding or receiving data.

7.5 Summary

This ARP-based anomaly detection approach has been designed to be deployed in a single *cell*. We tested the approach in a small network environment (nonetheless with a reasonable amount of normal network traffic). Obviously tests in larger and more diverse networks are required to fully exercise the approach; however, our testbed has provided substantial evidence of the practical viability of this approach. We have provided specific recommendations for the anomaly score parameters and settings, based on our network topology. Different network environments will likely require different parameter settings that can be determined during the training period.

The minimum sustained scanning rate constraint of one scan per minute was a limitation of our prototype and not the overall approach. The same approach could be used to detect worms or network scanning tools that scan even less frequently than this threshold at the expense of more memory and poorer results in terms of accuracy and false positives. The amount of internal network dark space in our testbed significantly influenced (positively) the performance of the detection technique.

Our detection system is anomaly-based and therefore has the ability to detect emerging worms. The prototype automatically calculates the required individual device statistics and can determine an appropriate network-specific alert threshold (r). We have developed a full implementation of our ARP-based approach in a software prototype that runs on commodity hardware. We plan to make this software available to the public.

Chapter 8

Summary and Future Directions

8.1 Summary of Research Contributions

Hypothesis 1 stated "to better defend the enterprise network, it is possible to design and deploy a suite of practical scanning detection techniques, that improve upon existing approaches, with acceptably low false postive/negative rates, and that are responsive within a very small number of scans (e.g., 1 to 3 scans - as low as a single scanning attempt)." We accept this hypothesis. We have developed three scanning detection techniques (i.e. exposure maps, DNS-based scanning detection, and ARP-based scanning detection) that can identify network scanning activity regardless of whether it originates from a local host in the internal network (i.e. L2L or L2R scanning) or it originates from a remote host external to the network (i.e. R2L scanning). These scanning detection techniques are anomaly-based and rely on behavioral signatures based on the observation that a scanning host exhibits anomalous behavior distinct from normal hosts. All three have been implemented either as fully functional stand-alone software implementations or as Bro policies. We plan to make these implementations available to others through standard means such as sourceforge.net. Our evaluation of these techniques involved the use of three network datasets, one derived from a small University network and network traces from a large enterprise network (i.e. the LBNL/ICSI network trace repository).

Network scanning detection techniques can both leverage and benefit from local knowledge obtained from the interactions of the hosts in an enterprise network and the perceived origin (i.e. internal or external to the network) of the scanning activity. All of our techniques exploit network-centric knowledge obtainable only within the local network.

Hypothesis 2 stated "it is possible to make novel use of address resolution protocols to detect malicious network activity, including some zero-day worms." We accept this hypothesis. We have developed two new techniques to detect scanning systems within the local network based on the anomalous behaviors they exhibit when using the ARP and DNS address resolution protocols. Our DNS-based scanning detection approach was developed to combat scanning worm propagation within an enterprise network. However, this approach is valid for most automated network scanning tools such as [21, 67, 66] as they share the same exploit methodology as scanning worms. Namely, their goal is to rapidly identify and compromise as many systems as possible. A typical configuration parameter for automated attack tools is a range of numerical IP addresses that they use to target their victims.

During evaluation, our DNS-based scanning detection prototype was successful in detecting scanning worm propagation as well as automated network scans directed at the Internal Departmental Network cell of our enterprise network. We have demonstrated that this network-based detection approach can be used to quickly and accurately detect L2R network scanning activity. Regardless of the scanning rate, the detection algorithm is able to detect network scanning activity in a single scanning attempt. It relies on a network service found in every network (i.e. DNS), and being anomaly-based, has the ability to detect emerging worms or new types of network scanning tools. We have developed a full implementation of our approach in a software prototype that runs on non-specialized commodity hardware.

The ARP-based scanning detection approach was also developed to combat scanning worm propagation within an enterprise network (i.e. L2L) but it too could be used to detect scans originating from most automated network scanning tools such as [21, 67, 66]. It is based on the observation that a scanning host targeting systems within its own network exhibits anomalous behavior distinct from normal ARP activity. We have provided specific recommendations for the anomaly score parameters and settings, based on our network topology. Different network environments will likely require different parameter settings that can be determined during the training period. The minimum sustained scanning rate constraint of one scan per minute was a limitation of our ARP-based detection prototype and not the overall approach. The same approach could be used to detect worms or network scanning tools that scan even less frequently than this threshold at the expense of more memory, poorer accuracy, and more false positives. The amount of internal network dark space in our testbed significantly influenced (positively) the performance of the detection technique. This technique is anomaly-based and therefore we would expect that it has the ability to detect emerging scanning worms or new types of network scanning tools. The prototype automatically calculates the required individual device statistics and can determine an appropriate network-specific alert threshold (r).

Hypothesis 3 stated "it is possible to devise a highly efficient scan detection technique which does not rely on who is doing the scanning, but rather on what service (or in general, what resource) is being scanned for. Here, as one example of efficiency, system state (in terms of main memory consumed) need not increase linearly with bursts in external scanning activity." We accept this hypothesis. Our exploration of this hypothesis involved the creation of a new R2L scanning detection technique (i.e. exposure maps) and the development of two heuristics to enable a side-by-side comparison with TRW (the exposure maps DCA heuristic) as well as provide the capability to detect distributed scanning activity. Additionally, we developed a modified TRW technique that uses all the internal logic (i.e. it makes use of a likelihood ratio and all the associated parameters) of the original TRW algorithm augmented with a NEM-based oracle.

Exposure maps differ from current scanning detection techniques [29, 58, 31, 64] as they rely on identifying the services offered by the network instead of tracking external connection events. Against a growing array of remote scanning techniques attribution (i.e. the identification of scanning systems) is becoming a quixotic approach to scan detection that overlooks an often critically important question that we suggest should be a much higher focus of scanning detection, namely, what is the adversary looking for? The basic exposure maps technique is based on a simple premise that is efficient to implement – it requires the passive observation, recording, and maintenance of a list of the services offered by the hosts in a network. When used as a scanning detection technique, the result is one in which the utilized system detection state does not grow in proportion to the amount and fluctuation of external network traffic, but rather increases only with the number of services offered by the network, regardless of the size of the network and the external network activity. This obviates the need for shrinking time windows or timeouts to accommodate increases or bursts in network traffic, allowing scan detection with a footprint of a single packet or a frequency of hours or days between probes. In contrast, for example, the TRW algorithm implemented in Bro [29] that we tested in Section 5.1.1 uses a 30 minute time interval to track and associate related failed connection attempts to identify potential scanning systems. Exposure maps make very efficient use of system detection state and computational resources easily scales for use in large enterprise and backbone networks. As an added benefit, maintaining information about internal hosts in the network instead of external host activity provides the necessary network-awareness to answer in real-time questions that should be asked after a scan is detected, such as "What information has been revealed as a result of the scan?", and "Has the network behavior changed?"

In an open network environment, the diversity of user population and permitted activity may make the enforcement of a single comprehensive network security policy impractical. Furthermore, mobile or transient users may make determining a stable baseline of all the services offered by hosts in a given network infeasible. An unvetted NEM was specifically designed for this type of network environment. An unvetted NEM is used when the permissive nature of the network allows the use of a variety of network services as part of the standard operating environment. The unvetted NEM, in effect, becomes the constantly updated corpus of active hosts and services found in the network. With the exception of the automated response capability, the unvetted NEM can enable all the capabilities associated with a vetted NEM (i.e. scanning detection, exposure profiles, and RAA). Alternately, in such diverse network environments a vetted NEM remains flexible enough to be configured to monitor a subset of the network to protect core network assets. For instance, a vetted NEM could be composed of a single HEM (e.g. primary web server or for host-based intrusion detection) or several HEMs (e.g., web server farm), allowing a network operator to focus on these mission critical servers.

Our exposure maps technique was evaluated using a side-by-side comparison with TRW and a modified TRW technique using a NEM-based oracle we developed (i.e. our evaluation consisted of only a few selected values of TRW parameters for the network datasets and thus it was a limited comparison; a more complete analysis would explore a full range of values). All of these techniques were tested with 3 network datasets. The exposure maps technique had the same performance in every individual test during the evaluation in terms of false positive and negative rates. An additional benefit of the modified TRW technique is that the NEM acts as a connection oracle that obviates the need for the algorithm to wait for connection responses in order to determine successes or failures. This could dramatically improve the performance of the TRW algorithm in terms of speed of detection.

The network-centric knowledge gathered by our exposure maps technique also offers the ability to identify potentially infected systems through the classification into exposure profiles based on the services they offer as well as provide a mechanism to create Reconnaissance Activity Assessments (RAA) identifying the network information divulged (i.e. hosts, open services, applications) to an adversary as a direct result of a specific scanning campaign.

The vetted NEM provides context to determine if an incoming connection request is part of a scanning campaign and whether it will likely elicit a response. This information provides the possibility of limiting containment to (e.g., automatically block) only those scanning systems targeting services offered by the network. Precise active response options can be restricted to the most critical known threats to the network; namely, those scanning systems targeting services offered by the network. Our testing with the CCSL dataset showed that only 8% of the detected scanning systems targeted a service offered by the network and should be blocked. This represents a 92% reduction in the number of dynamic updates to the network security ACLs if all scanning systems were candidates for an active response option.

8.2 Future Work

Our exposure maps technique produces DCAs that can be processed and analyzed using a variety of heuristics. For instance, we developed two heuristics to detect a form of distributed scanning. Additionally, we discussed a third heuristic that could be used to detect slow scanning. Other heuristics could be developed that use the raw output from exposure maps to identify other types of simple or sophisticated scanning activity (e.g. slow scanning).

The RAA provides a mechanism to identify the overall network information divulged as well as the specific network information revealed as a direct result of a specific scanning campaign. The signature file that enables RAA currently contains `regex` patterns for the SMTP, SSH, HTTP, DNS, and SSL protocols. Additional signatures could be developed to identify other types of applications that could be added into the signature file.

Our DNS-based detection technique is based on the observation that a local host performing remote network scanning does not perform a DNS-lookup before making a connection attempts. This behavior is inherently anomalous and we believe that it can be extended to detect additional classes of malicious activity including: covert channels, and remote to local worm propagation. The technique uses a whitelist to exempt a list of IP addresses and port combinations to eliminate a possible source of false positives caused by client applications that legitimately do not rely on DNS to function. For example, in most networks there are systems that regularly communicate with one another by using IP addresses specified in configuration files rather than fetched in DNS records. However, in certain network environments (e.g. ad hoc, P2P) the majority of hosts may initiate new connections without first generating a DNS lookup. In these network environments the number of IP addresses and protocols added to the whitelist may potentially limit the applicability of this technique as a stand alone detector. In these scenarios, an investigation could be undertaken of how this detection method could be used as an additional detection signal in concert with other worm detection schemes instead of being used as the primary detection technique.

Our experimentation has shown that the ARP-based detection technique has a greater speed of detection when the amount of network dark space is large in proportion to active hosts. The development of less coarse thresholds would be worth investigating when the size of network dark space is relatively small in order to increase both the speed and accuracy of the technique. One such method would be to evaluate how darkports could be used in conjunction with darkspace to detect L2L scanning activity.

Bibliography

[1] Optixpro trojan horse. http://securityresponse1.symantec.com/sarc/sarc.nsf/ html/backdoor.optixpro.12.html; accessed on January 24, 2008.

[2] tcpdump/libpcap public repository. http://www.tcpdump.org.

[3] M. Alsaleh, D. Barrera, and P. C. van Oorschot. Improving security visualization with exposure map filtering. In *Proceedings of the 24th Annual Computer Security Applications Conference (ACSAC)*, December 2008.

[4] G. Bakos and V. Berk. Early detection of Internet worm activity by metering ICMP destination unreachable activity. In *SPIE Conference on Sensors, and Command, Control, Communications and Intelligence*, April 2002.

[5] P. Barford and V. Yegneswaran. An Inside Look at Botnets. *Special Workshop on Malware Detection*, Advances in Information Security, Springer Verlag, 2006.

[6] G. Bartlett, J. Heidemann, and C. Papadopoulos. Understanding passive and active service discovery. In *IMC '07: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, pages 57–70, New York, NY, USA, 2007. ACM.

[7] R. Bejtlich. *Extrusion Detection, Security Monitoring for Internal Intrusions.* Addison Wesley, first edition, 2006.

[8] A. Bobyshev, P. DeMar, and D. Lamore. Effect of Dynamic ACL (Access Control List) Loading on Performance of Cisco Routers. In *Computing in High Energy Physics*, Feb. 2006.

[9] Bro Intrusion Detection System. http://bro-ids.org/; accessed on February 23, 2008.

[10] Reference Manual. Bro Wiki. http://www.bro-ids.org/wiki/index.php; accessed on February 23, 2008.

[11] S. Chen and Y. Tang. Slowing down internet worms. In *Proc. of 24th International Conference on Distributed Computing Systems*, Mar. 2004.

[12] S. Coull, C. Wright, F. Monrose, M. Collins, and M. Reiter. Playing devil's advocate: Inferring sensitive information from anonymized traces. In *Proc. of the 14th Annual Network and Distributed System Security Symposium*, Feb. 2007.

[13] M. Crispin. Internet Message Access Protocol. March 2003. http://www.ietf.org /rfc/rfc3501.txt?number=3501; accessed on Jan 12, 2008.

[14] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local worm detection using honeypots. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID'04)*, September 2004.

[15] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, February 2006.

[16] M. de Vivo, E. Carrasco, G. Isern, and G. O. de Vivo. A review of port scanning techniques. *SIGCOMM Comput. Commun. Rev.*, 29(2):41–48, 1999.

[17] R. Droms. Dynamic Host Resolution Protocol. RFC 2131, March 1997. http://www.ietf.org/ rfc/rfc2131.txt? number=2131; accessed on January 24, 2008.

[18] D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia. A behavioral approach to worm detection. In *Proceedings of The ACM Workshop on Rapid Malcode*, 2003.

[19] Forescout Technologies Inc, Forescout product. http://www.forescout.com /wormscout.html; accessed on January 24, 2008.

[20] F. C. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *ESORICS*, pages 319–335, 2005.

[21] Fyodor. Remote OS detection via TCP/IP stack fingerprinting. *Phrack*, 54, December 1998.

[22] C. Gates, J. J. McNutt, J. B. Kadane, and M. I. Kellner. Scan detection on very large networks using logistic regression modeling. In *ISCC '06: Proceedings of the 11th IEEE Symposium on Computers and Communications*, pages 402–408, Washington, DC, USA, 2006.

[23] G. Granger, G. Economou, and S. Bielski. Self-securing network interfaces: What, why and how. Technical report, Carnegie-Mellon University, CMU-CS-02-144, May 2002.

[24] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley. Worm detection, early warning and response based on local victim information. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC'04)*, pages 136–145, Washington, DC, USA, 2004.

[25] W. Harrop and G. Armitage. Greynets: a definition and evaluation of sparsely populated darknets. In *Proceedings of the 2005 ACM SIGCOMM Workshop on Mining Network Data*, pages 171–172, 2005.

[26] IANA TCP/IP Port Assignments. February 2008. http://www.iana.org/assignments/port-numbers; accessed on February 23, 2008.

[27] M. S. Johns. Identification Protocol. RFC 1413, February 1993. http://www.ietf.org/rfc/rfc1413.txt; accessed Jan 07, 2008.

[28] J. Jung. *Real-Time Detection of Malicious Network Activity Using Stochastic Models*. PhD thesis, Massachusetts Institute of Technology, 2006.

[29] J. Jung, V. Paxson, A. Berger, and H. Balakrishman. Fast portscan detection using sequential hypothesis testing. In *IEEE Symposium on Security and Privacy*, pages 211–225, 2004.

[30] C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for intrusion detection. Technical report, Technical University Vienna, Vienna, Austria. TU-1841-2002-28, 2002.

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

[32] B. Malmedal. Using netflows for slow portscan detection. Master's thesis, Department of Computer Science and Media Technology, Gjovik University College, Norway, 2005.

[33] MaxMind Frequently Asked Questions. *MaxMind GeoIP product.* http://www.maxmind.com/app/faq#accurate; accessed on January 24, 2008.

[34] J. McHugh. Acquisition and analysis of large scale network data: Introduction. Tutorial given at the 21st Annual Computer Security Applications Conference (ACSAC), December 2005.

[35] D. Mills. Network Time Protocol (Version 3). RFC 1305, March 1992. http://www.ietf.org/rfcs /rfc1305.txt?number=1305; accessed on January 24, 2008.

[36] Mirage Networks. Mirage NAC. http://www.mirage networks.com; accessed on January 26, 2008.

[37] P. Mockapetris. Domain Names - Implementation and Specification. RFC 1035, November 1987. http://www.ietf.org/rfcs /rfc1035.txt?number=1035; accessed on January 24, 2008.

[38] D. Moore. Network telescopes: Tracking denial-of-service attacks and Internet worms around the globe. In *LISA*, 2003.

[39] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the Slammer Worm. In *IEEE Security and Privacy Magazine*, pages 33–39, July/August 2003.

[40] D. Moore, G. Voelker, and S. Savage. Inferring Internet denial of service activity. In *10th USENIX Security Symposium*, 2001.

[41] Nessus. Tenable Network Security. http: //www.nessus.org.

[42] R. Pang, M. Allman, M. Bennett, J. Lee, V. Paxson, and B. Tierney. A first look at modern enterprise traffic. In *IMC'05: Proceedings of the Internet Measurement Conference 2005 on Internet Measurement Conference*, pages 2–2, Berkeley, CA, USA, 2005. USENIX Association.

[43] R. Pang, M. Allman, V. Paxson, and J. Lee. The devil and packet trace anonymization. *SIGCOMM Comput. Commun. Rev.*, 36(1):29–38, 2006.

[44] S. Panjwani, S. Tan, K. Jarrin, and M. Cukier. An Experimental Evaluation to Determine if Port Scans are Precursors to an Attack. In *International Conference on Dependable Systems and Networks*, pages 602–611, July 2005.

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

[46] V. Paxson, R. Pang, M. Allman, M. Bennett, J. Lee, and B. Tierney. LBNL/ICSI Enterprise Tracing Project (collection). `http://imdc.datcat.org/collection/1-0132-C=LBNL\%2FICSI+Enterprise+Tracing+Project;` accessed on February 12, 2008.

[47] R. Pethia. Attacks on the Internet 2003. *Congressional Testimony, Subcommittee on Telecommunications and the Internet*, USA, November, 2003.

[48] D. Plummer. An Ethernet Address Resolution Protocol 826. RFC 826, November 1982. http://www.ietf.org/rfc/rfc0826.txt?number= 826; accessed on January 24, 2007.

[49] J. Postel and J. Reynolds. File Transfer Protocol (FTP). RFC 959, October 1985. http://www.ietf.org/rfc/rfc959. txt?number=959; accessed Jan 02, 2006.

[50] N. Provos, D. McNamee, P. Mavrommatis, K. Wang, and N. Modadugu. The Ghost in The Browser: Analysis of Web-based Malware. In *USENIX HotBots 2007*, Aug. 2007.

[51] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Internet Measurement Conference 2006 (IMC'06)*, October 2006.

[52] Regular-Expressions.info. http://www.regular-expressions.info/; accessed on Feb 23, 2008.

[53] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear. Address Allocation for Private Internets. http://www.ietf.org/rfc/rfc1918.txt; accessed on Jan 02, 2006.

[54] RemoteScan Corporation, RemoteScan. http://www.remote-scan.com.

[55] D. Roelker, M. Norton, and J. Hewlett. sfPortscan. 2004. http://cvs.snort.org/viewcvs.cgi/snort/doc/README.sfportscan?rev=1.6.

[56] M. Roesch. Snort - lightweight intrusion detection for networks. In *LISA*, 1999.

[57] S. Sanfilippo. Bugtraq: new TCP scan method. December 1998. http://seclists.org/lists/bugtraq/1998/Dec/ 0079.html.

[58] S. Schechter, J. Jung, and A. Berger. Fast detection of scanning worm infections. In *7th International Symposium on Recent Advances in Intrusion Detection (RAID 2004)*, September 2004.

[59] R. Sekar, A. Gupta, J. Frullo, T. Shanbhag, A. Tiwari, H. Yang, and S. Zhou. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pages 265–274. ACM Press, 2002.

[60] C. Shannon and D. Moore. The spread of the Witty worm. *IEEE Security and Privacy*, 2(4):46–50, March 2004.

[61] M. Shelton. Passive Asset Detection System (PADS). http://passive. sourceforge.net; accessed on February 23, 2008.

[62] M. Smart, G. R. Malan, and F. Jahanian. Defeating TCP/IP stack fingerprinting. In *SSYM'00: Proceedings of the 9th USENIX Security Symposium*, pages 17–17, Berkeley, CA, USA, 2000. USENIX Association.

[63] A. Sridharan, T. Ye, and S. Bhattacharrya. Connectionless port scan detection on the backbone. In *Malware workshop, IPCCC*, Pheonix, AZ, April 2006.

[64] S. Staniford, J. Hoagland, and J. McAlerney. Practical automated detection of stealthy portscans. In *7th ACM Conference on Computer and Communications Security*, 2000.

[65] S. Staniford, V. Paxson, and N. Weaver. How to 0wn the internet in your spare time. In *Proceedings of the 11th USENIX Security Symposium*, August 2002.

[66] M. Tanase. Introduction to Autorooters: Crackers Working Smarter, not Harder. *SecurityFocus*, August 2002. http://www. securityfocus.com/infocus/1619.

[67] Tenable Network Security, Inc., Nessus Vulnerability Scanner. http://www.nessus.org; accessed on January 26, 2008.

[68] N. Weaver. Potential strategies for high speed active worms: A worst case analysis. 2002. http://www.cs.berkeley.edu/~nweaver/worms.pdf; last accessed October 20, 2004.

[69] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *The First ACM Workshop on Rapid Malcode*, Oct. 2003.

[70] N. Weaver, S. Staniford, and V. Paxson. Very fast containment of scanning worms. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[71] D. Whyte, E. Kranakis, and P. van Oorschot. DNS-based detection of scanning worms in an enterprise network. In *Proc. of the 12th Network and Distributed System Security Symposium*, Feb. 2005.

[72] D. Whyte, E. Kranakis, and P. C. van Oorschot. ARP-based detection of scanning worms in an enterprise network. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC)*, December 2005.

[73] D. Whyte, P. van Oorschot, and E. Kranakis. Exposure Maps: Removing Reliance on Attribution during Scanning Detection. In *USENIX HotSec 2006*, Aug. 2006.

[74] D. Whyte, P. C. van Oorschot, and E. Kranakis. Addressing SMTP-based mass-mailing activity within enterprise networks. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC)*, December 2006.

[75] D. Whyte, P. C. van Oorschot, and E. Kranakis. Tracking Darkports for Network Defence. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)*, December 2007.

[76] M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *18th Annual Computer Security Applications Conference (ACSAC)*, 2002.

[77] C. Wong, S. Bielski, A. Studer, and C. Wang. Empirical analysis of rate limiting mechanisms. In *8th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2005.

[78] A. Wool. A Quantitative Study of Firewall Configuration Errors. *IEEE Computer*, 37(6):62–67, 2004.

[79] V. Yegneswaran, P. Barford, and J. Ullrich. Intrusions: Global characteristics and prevalence. In *SIGMETRICS*, 2003.

[80] T. Ylonen. SSH – secure login connections over the internet. In *Proceedings of the 6th USENIX Security Symposium*, pages 37–42, 1996.

[81] M. Zalewski. p0f. http://lcamtuf.coredump.cx/p0f.shtml.

[82] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for Internet worms. In *Proceedings of the 10th ACM Conference on Computer and Communications Security*, 2003.

Appendix A

Supplementary Material

A.1 Acronym List

Table A.1: Acronyms.

ACE	ARP Correlation Engine
ACL	Access Control List
ARP	Address Resolution Protocol
CCSL	Carleton Computer Security Lab
DCA	Darkport Connection Attempt
DNSCE	DNS Correlation System
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name Service
HEM	Host Exposure Map
HTTP	Hypertext Transfer Text Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IDN	Internal Departmental Network
IMAP	Internet Message Access Protocol
IP	Internet Protocol
NEM	Network Exposure Map
L2L	Local to Local
L2R	Local to Remote
LBNL/ICSI	Lawrence Berkeley National Laboratory/International Computer Science Institute
MAC	Media Access Control
P2P	Peer to Peer
PPE	Packet Processing Engine
R2L	Remote to Local
RAA	Reconnaissance Activity Assessment
SMTP	Simple Mail Transfer Protocol
SSH	Secure Shell
SSL	Secure Sockets Layer
TCP/IP	Transmission Control Protocol/Internet Protocol
TRW	Threshold Random Walk
TTL	Time to Live
UDP	User Datagram Protocol

A.2 IANA Port Assignment List

Table A.2 contains the assigned port number to well known services according to the Internet Assigned Numbers Authority's (IANA) official TCP/IP port list [26].

Table A.2: IANA Assigned Port Numbers to Specific Network Services.

21/TCP	File Transfer Protocol (FTP)
22/TCP	Secure Shell (SSH)
23/TCP	Telnet
25/TCP	Simple Mail Transfer Protocol (SMTP)
53/UDP	Domain Name Transfer (DNS)
53/TCP	Domain Name Transfer (DNS)
80/TCP	Hyper Text Transport Protocol (HTTP)
110/TCP	Post Office Protocol (POP)
111/TCP	SUN Remote Procedure Call
111/UDP	SUN Remote Procedure Call
113/TCP	Authentication Service
119/TCP	Network News Transfer Protocol
135/TCP	DCE Endpoint Resolution
138/UDP	Netbios Datagram Service
143/TCP	Internet Message Access Protocol
161/UDP	Simple Network Management Protocol (SNMP)
427/UDP	Server Location
443/TCP	HTTP Protocol Over TLS/SSL
445/TCP	Microsoft-DS
515/TCP	Printer Spooler
554/TCP	Real Time Streaming Protocol (RTSP)
993/TCP	IMAP4 Protocol Over TLS/SSL
995/TCP	POP3 Protocol over TLS/SSL
1433/TCP	Microsoft SQL Server
1755/TCP	MS-Streaming
1863/TCP	MSNP
3389/TCP	MS WBT Server
5050/TCP	Multimedia Conference Control Tool
5061/TCP	SIP-TLS
7000/TCP	AFS3 Fileserver