Detecting Intra-enterprise Scanning Worms based on Address Resolution

David Whyte, Paul C. van Oorschot, Evangelos Kranakis
School of Computer Science
Carleton University, Ottawa, Canada
{dlwhyte, paulv, kranakis}@scs.carleton.ca

Abstract

Signature-based schemes for detecting Internet worms often fail on zero-day worms, and their ability to rapidly react to new threats is typically limited by the requirement of some form of human involvement to formulate updated attack signatures. We propose an anomaly-based detection technique detailing a method to detect propagation of scanning worms within individual network cells, thus protecting internal networks from infection by internal clients. Our software implementation indicates that this technique is both accurate and rapid enough to enable automatic containment and suppression of worm propagation within a network cell. Our approach relies on an aggregate anomaly score, derived from the correlation of Address Resolution Protocol (ARP) activity from individual network attached devices. Our preliminary analysis and prototype indicate that this technique can be used to rapidly detect zero-day worms within a very small number of scans.

1. Introduction

Scanning worms are rapidly evolving. Despite some very promising recent proposals (e.g. see [18]), most signature-based detection techniques are limited in their ability to detect newly emerging worms. Compounding the scanning worm detection challenge are the numerous worm variants typically launched after an initial outbreak. Slight code modifications enable worms to evade many signature-based countermeasures.

Worm infected hosts can potentially initiate thousands of infection scans per second [12] making automated suppression and containment strategies necessary [13]. The detection methods used within these automated schemes must be fast and accurate to ensure minimal impact to legitimate users. Most current worm propagation detection methods are limited in: (1) their speed of detection, (2) their inability to accurately detect *zero-day* worms, (3) their inability to detect slow scanning worms, and (4) their high false positive rate.

Containment within an enterprise network typically involves dividing the network into cells [4]. Although these cells can be as small as a single system (i.e. host-based containment) in a large network, the individual cells could be comprised of hundreds of hosts. The goal of such containment is to limit infection to individual cells; infection between hosts within a cell remains unchecked, with the potential loss of the hosts within a network cell considered an acceptable cost given the alternative of the entire network becoming infected. In this paper, we propose a detection technique that detects scanning worms attempting to propagate between hosts within an individual network cell. For the purposes of our approach, we define a *cell* as any portion of a network within a common broadcast domain.

We propose a behavioral signature[1] to detect scanning worms within an enterprise environment based on the observation that a scanning worm targeting systems within its own network cell exhibits anomalous behavior distinct from normal Address Resolution Protocol (ARP) [15] activity; infected systems trigger unusual ARP request activity as they try to infect susceptible systems within their respective network cells. More specifically, intra-cell worm-initiated scans result in discernible behavioral changes in the amount and pattern of ARP request activity of infected hosts, because a scanning worm targeting same-cell systems triggers the broadcast of anomalous ARP requests.

Our Contributions. We present a technique, which we have implemented and tested in a software prototype, to both rapidly and accurately detect worm propagation within enterprise network cells. Based on the following three factors, we derive an anomaly score for each individual device[2], and use this as an infection indicator for each device within a cell.

1. *Peer list*: connections to devices outside the set of internal devices a host normally interacts with.
2. *ARP activity*: increases in the average number of ARP requests each host issues per unit time.
3. *Internal network dark space* [1, 2]: connections to vacant internal IP addresses (i.e. addresses not bound to any active devices).

[1] Behavioral signatures [6] are used to describe common aspects of worm behavior particular to a given worm.

[2] Hereafter by *device*, we typically mean a network addressable host within a broadcast domain.

This approach offers significant performance improvement from the dynamic queues used by current worm scanning rate limiting techniques [22, 17] in discovering internal network (i.e. intra-cell) worm scanning (see further discussion in Section 2). Individually, each of these factors builds on previous work (e.g. [22, 14, 20]). However, we have combined them into a practical and tested enterprise detection technique using a layer 2 (i.e. ARP) protocol.

In our test environment, our technique detected simulated scanning worm activity with a minimum sustained scanning rate of one scan per minute, within three scans, with a very low false positive rate. The precision of this first-mile detection enables the use of automated containment and suppression strategies [13] to stop scanning worms before they infect other devices within a network cell.

Our anomaly-based detection approach is appealing for a number of reasons including the following:

1. *Speed*: the possibility to detect an infected system within e.g. three scans.
2. *Detection of zero-day worms*: possible because the approach does not rely on the matching of existing worm signatures to identify suspicious traffic.
3. *Scanning rate independence*: the approach can detect both fast and slow scanning worms (assuming a sustained scanning rate of at least one scan per minute - although this is configurable).
4. *Intra-cell protection*: our approach addresses infection of systems within a network cell.
5. *Low-false positive rate*: our prototype, albeit on a small network (see Section 5.1), experienced only five false positives within a two week period and this number could be further reduced (see Table 2).

Although certain marketing-related documentation offers the possibility that intra-cell worm propagation is addressed in at least one available commercial product [2] (albeit not involving ARP), we are not able to find any details from available published materials. To the best of our knowledge, our paper is the first in the open literature to propose a method and detailed description for, and to report on an implementation of, a detection technique that enables scanning-worm detection *within* a network cell (i.e. intra-cell propagation).[3] Used in conjunction with other internal scanning worm containment schemes that address propagation between network cells [20, 21, 22], our technique enables a complete enterprise-wide scanning worm detection capability (i.e. both intra-cell and inter-cell).

The sequel is structured as follows. Section 2 reviews related work. Section 3 outlines the basic approach. Section 4 discusses our prototype, which is analyzed in Section 5 with details for a particular dataset and small testbed. Section 6 presents limitations. We conclude in Section 7.

[3]Here, we assume a network cell contains more than a single device.

2 Related Work

The ARP protocol has been the subject of so-called *ARP cache poisoning* attacks [11], in which a device's ARP cache is updated with forged ARP request and reply packets in an effort to change the MAC address of the device to one in which an attacker can monitor or perform a denial of service attack. Other than the involvement of ARP, our work is unrelated to ARP cache poisoning.

A host-based solution to the scanning worm problem was discussed by Williamson [22]. His approach is to limit or throttle the rate of malicious code at the host by determining if a system is trying to connect to new addresses. If so, the connection is delayed in order to delay malicious code propagation. The decision to delay and not drop suspicious connections reduces the impact of false alarms while still limiting the spread of malicious activity in the network.

Schechter et al. [17] present a hybrid approach to detecting scanning worms using sequential hypothesis testing and rate limiting. A *first-contact* connection is defined as a connection attempt to a host that the system has not previously connected with. A finite number of these first-contact host addresses are kept in a queue and transition among three states: pending, success, or failure. When the queue is full and new addresses need to be added, the associated connections are subject to some form of performance penalty that effectively rate limits new connections from the host.

Certain aspects of our detection technique are similar to these rate limiting schemes. However, unlike dynamically allocated queues that record a transient connection history for a device, our queues are created during a training period and maintain a complete history of unique internal connections for each device.

Schemes that characterize internal to remote connections have to maintain a dynamic connection queue to accommodate the wide variance of host addresses the device may try to connect to during normal network activities (e.g. Internet web surfing). In contrast, we only need account for device connections to internal systems within the network. 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. This hypothesis is confirmed by our analysis in Section 5 (i.e. cf. Table 1, ARP Chain Size). For any given device, our model is that connecting to devices with which they have prior connection history should occur at a higher probability than connections to devices with which they have no connection history.

Ganger et al. [8] developed a software architecture to enable self-securing network interfaces. Packets are examined as they are processed by host software where malicious activity can be detected and potentially blocked.

Whyte et al. [21] used DNS activity to detect the presence of scanning worms within an enterprise network. The observation of connections outside the network not preceded by a DNS query was considered anomalous and a strong indicator of scanning worm activity.

Jung et al. [10] developed an algorithm called Threshold Random Walk (TRW), to identify malicious remote hosts, based on the observation that scanners are more likely to attempt to access hosts and services that do not exist than legitimate remote hosts. Weaver et al. [20] developed a scan detection and suppression algorithm based on a simplification of TRW. They use caches to track the activity of both new connections and IP addresses to reduce the random walk calculation, making the algorithm suitable for implementation in both hardware and software.

Zou et al. [23] model requirements for the dynamic quarantine of infected hosts. They demonstrate that epidemic thresholds exist for differing detection and response times. This work provides a benchmark that any new proposed detection algorithm could use to measure its efficiency. Singh et al. [18] propose a method to automatically detect worm propagation based on network traffic characteristics such as highly repetitive packet content, and IP source and destination address distributions. They have shown that this method has the ability, with no human intervention, to generate worm content signatures with a low false positive rate.

Two commercial scanning detectors offer an alternate approach to detect worm propagation. Forescout [1] and Mirage [2] networks use a technique called *dark-address detection*. These detectors either have knowledge of, or route unoccupied address spaces within an internal network and detect when systems attempt to connect to unused spaces. A variety of network-based solutions to the scanning problem exist including Silicon Defense's *CounterMalice* [4] which is a worm defense solution that proactively identifies and automatically blocks worm activity within a network. The solution partitions the network into cells and prevents worms from spreading between the cells. Moore et al. [14] describe how large portions of routed IP address space with little or no legitimate traffic can be turned into a *network telescope*. The monitoring and analyzing of unexpected traffic sent to a network telescope provides opportunities to view Internet-wide security events (e.g. denial of service attacks, Internet worms).

3 Basic Methodology and Approach

In this section we give a high-level overview of our ARP-based scanning worm anomaly detection approach.

In larger enterprise networks, it is not unusual for network segments to be either logically or physically separated. This natural separation of networks can be leveraged to contain worm propagation within distinct network segments or *cells*.

As has been noted elsewhere (e.g. see [21]), the propagation of scanning worms can be characterized as: local to local (L2L), local to remote (L2R), or remote to local (R2L). In L2L propagation, the worm targets systems within the boundaries of the enterprise network in which it resides e.g. scanning within network cells (i.e. *intra-cell*) or between them (i.e. *inter-cell*).[4] L2R propagation refers to a scanning worm within an enterprise network targeting systems outside of its network boundary. Finally, R2L propagation refers to worm scanning from the Internet into an enterprise network. Our approach addresses L2L worm propagation within a network cell (i.e. L2L intra-cell); it does not address L2L inter-cell, L2R, or R2L worm propagation.

3.1 ARP Anomaly 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 request. The protocol stipulates 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 associated IP and MAC. ARP was designed as the intermediary between IP and MAC addresses within local subnets. Other technologies such as Windows Internet Naming Service (WINS), Internet Protocol (IP), and Domain Name Service (DNS) that enable communication outside of local subnets all rely on ARP.

For the purposes of our approach, we define a *scan* as an ARP request. An ARP request indicates that a system is trying to resolve an IP address to a MAC address for some type of connection. 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 3.1.2. 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.

[4] Topological scanning worms (e.g. Nimda) employ this strategy [19].

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 these behaviors to derive an aggregate anomaly score for each device within a cell.

3.1.1 Peer List (Customary ARP request targets)

Each time an ARP request is generated, any observed new source protocol address is recorded as an index entry within 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 requestors) 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 requestor. If an ARP requestor 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.).[5]

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. Subsequent distinct connection attempts outside a device's ARP chain within the detection window (see Section 3.2) result in a linear growth for this anomaly factor.

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, let $a_{1,i}^{(j)}$ denote the anomaly score $a_{1,i}$ as defined above at sample interval j; then the running total for $a_{1,i}$ for the current detection window of width w is $A_{1,i} = \sum_{k=0}^{w-1} a_{1,i}^{(j-k)}$.

3.1.2 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. 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: $E_i = \overline{x} + 2\sigma$.

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:

$$a_{2,i} = \begin{cases} O_i - E_i & ;\text{if } \mathrm{O_i} > \mathrm{E_i} \\ 0 & ;\text{if } \mathrm{O_i} \leq \mathrm{E_i} \end{cases} \quad (1)$$

Similar to the a_1 factor, this calculation is performed during each sample interval to determine a *running* total within the *detection window* (see Section 3.2) 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 $\overline{x} = 1.579$, $\sigma = 1.036$, and $E_{11} = 3.651$. If $O_{11} = 5$ then $a_{2,11} = 1.349$. This factor is particularly useful in addressing the affect of local ARP caches and large ARP chains (as discussed in Section 5.3.2).

3.1.3 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. ARP requests for IP addresses not contained within this set we consider to be anomalous, and refer to as *internal network dark space*.

We assign an anomaly score for the *internal network dark space factor* (a_3) during a given sample interval as:

$$a_{3,i} = \begin{cases} 0 & ;\text{if no dark space scans} \\ y & ;\text{if device i scans dark space} \end{cases} \quad (2)$$

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 Section 3.2).

3.2 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

[5]This assumption is violated e.g. in P2P networks and highly distributed cooperative network environments (see Section 6).

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: $a_{T,i} = a_{1,i} + a_{2,i} + a_{3,i}$.

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

$$A_i = \sum_{k=0}^{w-1} a_{T,i}^{(j-k)} \quad (3)$$

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. 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 1), 192.168.1.11 has the highest E_i score (3.651) therefore $r = 3$.

4 High-Level System Design

Our software implementation uses the *libpcap* [3] 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 (see Section 4.1). 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 (see Section 4.2). The ACE maintains in state all relevant ARP information extracted by the PPE.

4.1 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.

4.2 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, 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). If within a sampling interval there is no ARP request activity, this observation is excluded from final mean and standard deviation calculations (see Section 3.1.2). 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 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. 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.

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 3.1). 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.

5 Prototype and Analysis

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. Four weeks of network traffic was collected in one of our university research labs. The first two weeks of the network data set was used as the training period. We then tested the

Table 1. ARP statistics for prototype system. Mean and Standard Deviation for number of ARP requests made per sample interval of 60 seconds during the training period.

Servers				
Address	ARP Chain Size	$\overline{x}$	σ	Max Requests
192.168.1.11	21	1.579	1.036	8
192.168.1.12	17	1.203	0.610	9
192.168.1.13	16	1.176	0.428	4
Workstations				
Address	ARP Chain Size	$\overline{x}$	σ	Max Requests
192.168.1.16	10	1.243	0.467	4
192.168.1.20	9	1.214	0.467	5
192.168.1.24	8	1.171	0.423	4
192.168.1.26	11	1.068	0.261	3
192.168.1.27	8	1.197	0.470	4
192.168.1.30	5	1.724	0.674	5
192.168.1.31	4	1.522	0.639	3
192.168.1.33	6	1.235	0.449	3

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:

1. Common threshold approach: give every device within the network cell the same alert threshold r.
2. Function-specific threshold approach: 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.

5.1 Data Set for Prototype Evaluation

To validate our approach, we developed and tested a fully functional software prototype with all features discussed in Sections 4.1 and 4.2. The software was installed on a commodity PC running Linux with a 10/100 network interface card. The lab network consisted of a one quarter Class C network of Internet-reachable IPv4 addresses. Using the cell definition from Section 3, the lab contained one cell. 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 (i.e. Max Requests) within the 60 second sampling interval. 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 discussed in Section 5.3.1. 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 internal network independently with an intrusion detection system (i.e. snort [16]) to ensure no known worm activity was included within the data set. Finally, we simulated scanning worm propagation within the test network using the *Nmap* [7] security scanner to test our detection software.

Table 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

The respective ARP request activity for a sample of active system within the internal network is included in Table 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.

Approach 1: Common Threshold. Table 1 separates servers and workstations in our testbed. Note that 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 network. This is not unexpected in a typical client-server model. Likewise, the servers within the 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 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 Section 3.2), 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 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 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. However, as r increases so does the number of worm scans before an alarm is generated.

5.2 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) 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 1). We set our alert threshold to $r = 3$. 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* (i.e. no time delays between scans) 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 for stealth) employing both the sequential and random scanning strategies.

Table 3. Network ARP statistics

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

5.2.1 Worm Simulation Results

Nmap Sequential Scanning Strategy. The sequential scans for both normal and sneaky mode were detected within two port scans (see Table 3). Nmap was configured to sequentially scan the host range from 192.168.1.1 to 192.168.1.62 (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.

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 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.

5.2.2 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 [19]. 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 (see Section 5.2.1).

5.3 Discussion of False Positives and Negatives

The following two sections 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).

5.3.1 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.

5.3.2 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 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 Section 3.1.2) does offer some insight on the affect of ARP caches. For example, system 192.168.1.30 (statistically the most active workstation in Table 2) has $\overline{x} = 1.724$ and $\sigma = 0.674$, making $E_{30} = 3.072$. 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 4.2) 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 $a_{2,i}$.

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

Table 4. Anomaly factor triggering probabilities in testbed

System Specific ARP Request Statistics			
	ARP Chain Size	$\overline{x}$	σ
192.168.1.11	21	1.579	1.036
192.168.1.26	11	1.068	0.261
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

chain (see discussion in 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 4 shows the probabilities of the two systems scanning within their respective ARP chains, internal network dark space, and their peer lists.

As discussed in Section 5.2.2, 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 5.3.2). 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 our network as a practical example, a sequential scan from 192.168.1.11 would deviate from its ARP chain on the second scan 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.

6 Limitations

Limitations. 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. 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 correlation would also address another potential limitation that arises in networks that use the Dynamic Host Configuration Protocol (DHCP) [5]. 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 (see Section 3.1.1). 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 dynamically 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.

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. This would require our prototype to observe a greater number of worm scans before the a_2 factor would trigger. In such instances, our technique could be integrated as a secondary input to a more sophisticated suite of anomaly detectors.

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. However, once propagation begins the worm's sustained scanning rate would still trigger an alert by a_2 unless the scanning rate was less than 1 scan per minute.

7 Concluding Remarks

The main objectives of this paper have been to describe an ARP-based anomaly detection approach designed to be deployed in a single *cell*, and confirm that the general idea works. We tested the approach in a small lab environment (nonetheless with a reasonable amount of normal network traffic) and provided evidence of the viability of this approach. 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 that scan less frequently than this threshold at the expense of more memory and poorer results in terms of accuracy and false positives.

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.

Acknowledgements

We thank John Ioannidis, Carleton University's Digital Security Group, and the anonymous reviewers for comments which significantly improved this paper. The second author is the Canada Research Chair in Network and Software Security, and is supported in part by an NSERC Discovery Grant, the Canada Research Chairs Program, and MITACS. The third author is supported in part by NSERC (Natural Sciences and Engineering Research Council of Canada) and MITACS (Mathematics of Information Technology and Complex Systems) grants.

References

[1] Forescout. Wormscout. http://www.forescout.com/ wormscout.html.

[2] Mirage Networks. http://www.miragenetworks.com.

[3] tcpdump/libpcap repository. http://www.tcpdump.org.

[4] Silicon Defense. Worm containment in the internal network. Technical report, 2003.

[5] R. Droms. Dynamic Host Resolution Protocol. RFC, March 1997. http://www.ietf.org/rfc/rfc2131.txt? number=2131; accessed on January 24, 2005.

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

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

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

[9] A. Gupta and R. Sekar. An approach for detecting self-propagating email using anomaly detection. In *Proceedings of the Sixth International Symposium on Recent Advances in Intrusion Detection*, September 2003.

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

[11] S. Manwani. ARP cache poisoning prevention and detection. Technical report, Faculty of Computer Science, San Jose State University, December 2003.

[12] 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.

[13] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proc. of the 2003 IEEE Infocom Conference, San Francisco, CA*, April 2003.

[14] D. Moore, C. Shannon, G. Voelker, and S. Savage. Network Telescopes. Technical report, CAIDA, April 2004.

[15] D. Plummer. An Ethernet Address Resolution Protocol. RFC, November 1982. http://www.ietf.org/rfc/rfc0826.txt? number=826; accessed on January 24, 2005.

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

[17] S. Schechter, J. Jung, and A. W. Berger. Fast Detection of Scanning Worm Infections. In *Proceedings of the Seventh International Symposium on Recent Advances in Intrusion Detection*, French Riviera, France, September 2004.

[18] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *ACM/USENIX Symposium on Operating System Design and Implementation*, Dec. 2004.

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

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

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

[22] M. Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Annual Computer Security Applications Conference*, 2002.

[23] C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and early warning for Internet worms. In *Proceedings of the 10th ACM CCS*, 2003.