DNS-based Detection of Scanning Worms in an Enterprise Network

David Whyte[†] Evangelos Kranakis[†] P.C. van Oorschot[†]

August 24, 2004

Abstract

Worms are arguably the most serious security threat facing the Internet. Motivated to develop a detection technique that is both efficient and accurate enough to enable automatic containment of worm propagation at the network egress points, we propose a new technique for the rapid detection of worm propagation from an enterprise network. Implemented in software, it relies on the correlation of Domain Name System (DNS) queries with outgoing connections from an enterprise network. Significant improvement over existing scanning worm detection techniques includes: (1) the possibility to detect worm propagation after only a single infection attempt; (2) the capacity to detect zero-day worms; and (3) a low false positive rate. The precision of this first-mile detection technique supports the use of automated containment and suppression strategies to stop fast scanning worms before they leave the network boundary. Furthermore, we believe that this technique can be applied with the same precision to identify other forms of malicious behavior within an enterprise network such as: mass-mailing worms, network reconnaissance activity, and covert communications.

1 Introduction

Recently, a multitude of high profile worm epidemics has affected millions of networked computing devices. The Slammer Worm that emerged in January of 2003 exposed how quickly worm propagation could occur, infected systems by exploiting a buffer overflow vulnerability in Microsoft SQL Server. Slammer's infected population doubled in size every 8.5 seconds [8] with 90% of vulnerable hosts infected in just 10 minutes. This worm achieved its full scanning rate (i.e. over 55 million scans per second) only 3 minutes after it was released. In August 2003 the SoBig worm caused an estimated $5 billion in damage and at the height of its infection was responsible for approximately 73% of all Internet email traffic [10]. Unfortunately, worm outbreaks of this scale are becoming commonplace. In March 2004, the Witty Worm began to spread by exploiting a buffer overflow in Internet Security Systems (ISS) products that include firewalls and intrusion detection systems. Although the vulnerable population of Internet systems was a magnitude smaller than previous worms, it spread very rapidly [12]. To achieve the rate of propagation observed, it is believed that the authors of this worm used a preprogrammed hitlist or a timed release of the worm on previously compromised systems. Witty was the first widely propagated worm to contain a malicious payload and signifies a disturbing new trend for worm writers, combining skill and malice [15].

Staniford et al. [13] hypothesize that a properly constructed worm could infect vulnerable systems on the Internet at an even greater speed. Worms are evolving and they can employ a number of anti-detection techniques such as: anti-forensics, dynamic behavior, and modularity of attack tools [11]. Furthermore, worms spread so quickly that traditional intrusion detection methods (i.e. generation and deployment of attack signatures) are not feasible [9]. In order to make automatic containment of fast scanning worms feasible, a rapid and accurate detection method is required.

Currently, most countermeasures used to mitigate these attacks include some form of human intervention. Routers can be configured to block network traffic and vulnerable software can be patched. However, worms that propagate and infect the Internet in just minutes make these human-in-the-loop countermeasures impractical. The development of wide scale automated countermeasures is required. Current worm propagation

[2]{dlwhyte, kranakis, paulv}@scs.carleton.ca. School of Computer Science, Carleton University, Ottawa, Canada.

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.

Typically, fast scanning worms use a pseudo random number generator (PRNG) to generate 32 bit random numbers that correspond to an IPv4 address. The attacking system uses this numeric address as the target for its infection attempt. 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. In contrast, the vast majority of legitimate publicly available services are accessed through the use the DNS protocol which provides the mapping between numeric IP addresses and the corresponding alphanumeric names. Most users find it hard to remember numeric addresses; a string of letters in the form a mnemonic is much easier. The translation of a host name to a registered IP address is called *resolving*. While there exists valid exceptions to this observation (e.g. client to client applications, remote administration tools, etc.) typical user behavior should include some form of DNS activity before a new connection can be initiated.

Our Contributions. We use DNS anomalies to detect scanning worm propagation, relying on the observation of DNS responses. If we do not observe DNS activity before a new connection is initiated, we consider this connection anomalous. This premise is based on our observation that automated worms behave differently than a normal user. Users tend to remember alphanumeric strings and use the network services provided to them (i.e. DNS). Almost all worms consume numeric IP addresses and have no such regard for using the network they infect in a standard way. We present a new technique, implemented and tested in a software prototype, to both rapidly and accurately detect worm propagation in an enterprise network. 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 use of our DNS anomaly-based detection approach for scanning worm propagation is appealing for a number of reasons including:

1. Speed: the possibility to detect an infected system after only a *single* infection attempt to the Internet.
2. Detection of zero-day worms: possible because our approach does not rely on the matching of existing worm signatures to identify suspicious traffic.
3. Scanning rate independence: our approach can detect both fast and slow (i.e. stealth) scanning worms.
4. Reduced training period: our approach includes the concept of a whitelist that can be quickly generated to reduce false positives.
5. Low-false positive rate: our approach does not rely on modeling normal network and user behavior profiles that are prone to false positives.
6. Ease of implementation: our approach is network-based, runs on commodity hardware, and relies on the observation of a protocol found in every network (i.e. DNS).

Our detection technique can be used to detect scanning worm propagation both within an enterprise network and from the enterprise network to the Internet (i.e. local to remote). It does not detect worm propagation from the Internet to the enterprise network. The DNS-approach to scan detection differs from existing scanning worm detection techniques in that it does not rely on having to observe and correlate multiple events in order to make a determination that a scan is occuring. There is no concept of a threhold; we only maintain in state a list of IP addresses of *valid* connection destinations and each individual connection attempt from the enterprise network as it occurs. Our DNS-based approach enables the detection of an infected system after a single scan has been initiated regardless of the time between scans. Our technique compares very favorably to previous work (e.g. Weaver et al. [16]). Weaver et al. propose a scanning detection algorithm based on the Threshold Random Walk (TRW) scan detector [6] to detect a scanning host within an enterprise environment after only 10 scans. Their detection algorithm can detect scans as slow as 1 scan per minute.

The sequel is structured as follows. Section 2 presents the description of the DNS-based scanning worm propagation detection technique. Section 3 discusses our experimental platform. Section 4 discusses the analysis of our prototype. Section 5 presents detection circumvention and limitations. Section 6 discusses ideas for extended applications of our detection technique. Section 7 discusses related work. We conclude in Section 8 with a brief summary. The Appendices contain background information and supplemental tables.

2 Basic Methodolgy and Approach

For an overview of worm propagation strategies and DNS please refer to Appendix A. In this section we give a high-level overview of our DNS-based anomaly scanning worm detection approach. The propagation of fast-scanning worms can be characterized as: local to local (L2L), local to remote (L2R), or remote to local (R2L). In L2L propagation, a scanning worm targets systems within the boundaries of the enterprise network it resides. Topological scanning worms employ this strategy. 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. We propose a worm propagation detection method to detect L2R worm propagation and worm propagation between cells. The detection technique does not address R2L worm propagation or worm propagation that occurs within an individual cell.

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 for a variety of reasons including security, ease of administration, and geographical location. We can leverage this natural separation of networks to contain L2L worm propagation within distinct network segments. Silicone Defense's CounterMalice solution [4] purposely divides the enterprise network into segments called *cells*. Each cell contains a worm containment device to confine and contain worm infection. Our definition of a cell refers to all systems within the same subnet serviced by a distinct authoritative DNS server. Figure 3 in Appendix E illustrates how an enterprise network can be divided into cells. Systems that reside within the same cell typically do not use DNS to communicate. The Address Resolution Protocol (ARP) [1] is used when a system tries to communicate with another system in the same cell. ARP is used by the data link layer to provide a mapping between the physical hardware of a system and its assigned IP address. L2L worm propagation can occur within a particular cell or span multiple cells depending on the scanning strategy of the worm.

DNS Anomaly 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 worm propagation. The technique does not detect R2L or intra-cell (i.e. within the boundaries of a cell) worm propagation.

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. DNS, 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 3.2), we consider any system that tries to access another system without receiving a valid DNS response as a possible worm infected system.

3 High-Level System Design

Our software system design uses the *libpcap* [3] library and is comprised of two logical components: PPE and DCE. The Packet Processing Engine (PPE) is responsible for extracting the relevant features from the live network activity or saved network trace files (see Section 3.1). The DNS correlation engine (DCE) maintains in state all relevant DNS information, a whitelist, and numeric IP addresses embedded in HTTP packets extracted by the PPE (see Section 3.2). 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* [3] 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 (see Figure 4 in Appendix E).

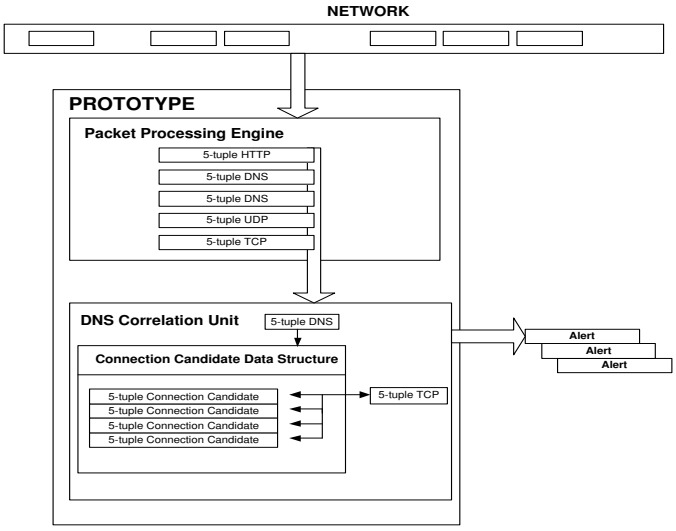

Figure 1: High-level System Design

Figure 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 (see Section 3.1) based on the protocol extracted. These tokens are consumed by the DCE. The DCE 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 DCE will generate an alert if it determines the new connection is being initated to a destination IP that is not contained in its list.

3.1 Packet Processing Engine

The PPE is responsible for extracting 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.

3.2 DNS Correlation Engine

The DNS correlation engine (DCE) is responsible for processing information passed by the PPE. The two major functions of the DCE 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 DCE'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 in 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 by HTTP 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. 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 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. Regardless, in many networks end users are restricted to using a finite list of well known services deemed permissible to the security policy of the network. For instance, it may be permissable 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, users could be told to enter in domain names instead of the IP address.

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.

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 DCE. 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 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. A sample of the alert output generated by our system is included below.

2004-6-25 12:43:09 TCP Connection 192.168.1.2:3456 - 192.168.200.1:80

The alert contains the time the activity was detected, protocol, source and destination IP address and source and destination port number.

4 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 3. To conduct our evaluation, one week of network traffic was collected at a firewall in front of one of our university's research labs. 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 lab router is connected to the university's Internet accessible Class B network. The lab network consists of a one quarter Class C network (i.e. 63) of Internet reachable IPv4 addresses.

The lab network contains one authoritative DNS server that all internal systems in the network are configured to use. The lab's DNS server has entries associated with the lab's mail server, web server, and Kerberos server. The firewall does not permit any inbound connections unless they were first established by an internal system. All systems within the lab 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 lab can be considered one cell in the university's enterprise network. The lab 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 lab 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 lab 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 lab 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 lab DNS server's cache. This ensured that any new connections from lab systems would result in an external DNS query to retrieve the appropriate A record instead of accessing the lab DNS server's cache. From our vantage at the network boundary, we are only able to detect DNS replies as they enter the lab network, not those generated internally from the DNS server's cache. The flushing of

Table 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 2: DNS Datagrams

Date	Total Packets	DNS Request Data-grams	DNS Reply Data-grams
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

the lab DNS cache ensures that the DCE will contain the same DNS information as the lab'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.

4.1 Data Set

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 lab'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 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 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.

4.2 Lab Monitoring Analysis

The lab 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. Appendix B contains Table 6 listing the seven entries that comprised the lab'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 lab 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 3 gives the alert breakdown by day. None of the alerts could be attributed to worm propagation but in contrast, see Section 4.3. This is not surprising since the lab 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 4.4.

4.3 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 7 in Appendix B). All of these entries consisted of allowed communications between specific servers. Over the one-week period, 74,610 alerts were generated as

Table 3: Lab Alerts

Date	# of Alerts	Known False Positives	True False Positives
06-24-2004	18	6 Internal Lab, 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

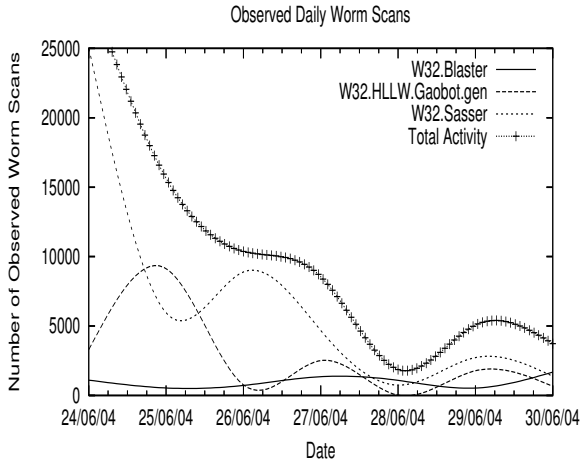

Figure 2: Worm Activity

a result of worm propagation attempts from the IDN to the lab. Table 8 in Appendix C 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 W32.HLLW.Gaobot.gen. We estimate that these worms infect a total of 195 IDN hosts. Figure 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 4 reveals the number and type of alerts generated. A full analysis of the false

Table 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	

positives generated by the prototype is given in section 4.4.

4.4 False Positives and False Negatives

False Positives Results Analysis

52 false alerts were generated from monitoring the lab 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:

1. Authorized network communications that could be incorporated into a whitelist but were not in our prototype testing.
2. Network configuration errors that could be eliminated with proper system administration.
3. Suspicious scanning activity other than worm propagation.
4. 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.

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 [2] (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.

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

After further analysis of the lab 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 5 summarizes the number and type of false positives generated from monitoring the lab and IDN cells.

Table 5: False Positive Results Analysis

	Lab	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 lab) 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.

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 candidate connection list. If the exchange of datagrams between the two systems is longer than the TTL of the DNS resource record 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 candidate connection data structure. That is, the intended victim was previously accessed by a system in the cell and the associated entry in the candidate connection 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 candidate connection data structure, the probability that a scan from the internal network will be directed at one of the data structures entries

$$\beta = r\frac{N}{2^{32}}. \tag{1}$$

For example, with a 10,000 entry candidate connection 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 candidate connection list is only .04%. Table 9 in Appendix D lists, for various parameters, the probability of a false negative occurring due to a previously cached IP address in the candidate connection data structure. insufficent observation period to generate

5 Detection Circumvention and Limitations

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 dection 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:

1. Require worms writers to adopt a new infection paradigm to randomly generate valid domain names instead of numeric IP addresses for targeting.
2. Slow propagation as worms perform DNS queries in order to spread.
3. Increase DNS activity as every infection attempt will pose a significant and noticeable impact on the DNS server.
4. Reduce the number of *reachable* vulnerable systems because not all systems (e.g. home users, client systems in an enterprise networks) have qualified domain 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).

Our DNS-based worm detection approach has other limitations. It cannot detect either R2L worm propagation 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. Finally, worm propagation via email/network share traversal will not be detected.

6 Extended Applications

Our DNS-based detection approach could be applied to four 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; and (4) mass mailing worm detection. 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 wormâs 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 the DNS server 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 detection technique can be developed to identify a mass-mailing worm infected system in a single propagation attempt.

7 Related Work

The observation of network service use, such as DNS, offers a means to detect anomalous network activity. Kruegel et al. [7] orginally 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 essiential 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.

Granger et al. [5] 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 worm propagation. The technique involves shadowing a host´s DNS table and checking the IP address of each new connection against it. The basic construct of this approach is that its abnormal for a host to make a large number connection attempts without DNS activity. We have extended it to a network-based solution that incorporates additional network information (i.e. whitelists, embedded IP addresses in HTTP packets) that hosts use to initiate new connections.

Williamson [17] devised a method to limit or throttle the rate of malicious mobile code by determining if a system trying to connect to new addresses. If so, the connection is delayed in order to slow the propagation of the malicious code. By not dropping the connection, a balance is struck between reducing the impact of false alarms and limiting the spread of malicious activity in the network.

Jung et al. [6] developed an algorithm called Threshold Random Walk (TRW), to identify malicious remote hosts. They based this algorithm on the observation that scanners are more likely to access hosts and services that do not exist than legitimate remote hosts. If the connection is determined to succeed the random walk is driven upwards, failure to succeed drives the random walk down. By giving legitimate network traffic a higher probability to succeed than attack traffic, a determination can be made on whether a series of connection attempts is a scan.

Weaver et al. [16] developed a scan detection and suppression algorithm based on a simplification of the TRW. They use caches to track the activity of both new connections and IP addresses to reduce the random walk calculation in the TRW. This simplication made the algorithm suitable for implementation in both hardware and software. Their technique allows a scanning host to be detected and stopped in fewer than 10 scans with a low false positive rate.

Silicon Defense developed the CounterMalice worm defense solution [4] to proactively identify and automatically block worm activity in an internal network. The solution divides the network in cells and prevents the worms from spreading between the cells.

Zou et al. [18] 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 against which the efficiency of any new proposed detection algorithm should take into account. We believe that our scanning worm detection approach has the required efficiency to stop worm propagation before epidemic thresholds are reached.

8 Concluding Remarks

The DNS-based worm propagation detection approach is an effective way to combat scanning worm infection within an enterprise network. During our evaluation, our prototype was successful in detecting worm propagation in the IDN cell of our enterprise network. We have demonstrated that this network-based detection approach offers a significant improvement in detection speed over existing scanning worm propagation detection methods. Regardless of the scanning rate, the detection algorithm is able to detect scanning worm propagaton in a single scanning attempt. The detection algorithm relies on a network service found in every network (i.e. DNS) and because it is anomaly-based, it has the ability to detect emerging worms. We have developed a full implementation of our DNS-based detection approach in a software prototype that runs on non-specialized commodity hardware. We plan to make the software available to the public.

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 R2L worm propagation.

References

[1] Address resolution protocol. http://www.faqs.org/rfcs/rfc826.html.

[2] Optixpro trojan horse. http://securityresponse1.symantec.com/sarc/sarc.nsf/html/backdoor.optixpro.12.html.

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

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

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

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

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

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

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

[10] T. News. August was worst month ever for viruses, worms. September 2003.

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

[12] C. Shannon and D. Moore. The spread of the witty worm. Technical report, CAIDA, March 2004.

[13] 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.

[14] N. Weaver. Potential strategies for high speed active worms: A worst case analysis.

[15] N. Weaver and D. Ellis. Reflections on witty. *;login: The USENIX Magazine*, 29(3):34–37, June 2004.

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

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

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

A Background

A.1 Worm Propagation Methods

Worms are typically classified based on two attributes: methods used to spread and the techniques used to exploit vulnerabilities. Most worms propagate by using indiscriminate scanning of the Internet to identify vulnerable systems. As revealed by Slammer, the faster a worm can locate systems the more rapid the infection rate. Staniford et al.'s study [13] used empirical data from actual worm outbreaks to reveal a common effective propagation strategy, random constant spread (RCS) model, wherein a worm randomly scanning through the entire Internet 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: (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) their resistance to countermeasures [14]. The spread of a random scanning worm can be described in three phases: the slow spreading phase, fast spreading phase, and slow finishing phase [18].

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 [13]. Worms use a number of different scanning strategies to propagate. Scans can be focused on specific groups of systems (e.g. subnet scanning) that are phyically or logically connected together. Some scanning strategies use information such as URL caches, peer-to-peer connections, trusted network connections, and email addresses harvested from their victims (e.g. topological scanning) to target potentially susceptible hosts [13].

In addition to scanning, worms have also used mass email and network shares to propagate. Using built-in emailers, worms can harvest 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 harvested email entries. Through shared network drives, systems often have access to directories on other systems. By placing itself in a shared system, the 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.

Hyper Virulent Worm Propagation Strategies. Weaver et al. [13] describe a number of possible hyper virulent worm propagation strategies that includes hit-list and permutation scanning. A hit-list is a list of vulnerable systems targeted for infection. Typically, a hit-list is generated by previous reconnaissance activities such as: network scanning, web surveys, DNS queries, 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. Permutation scanning is a strategy to increase scanning efficiency. Random scanning can be inefficient because many addresses may be probed multiple times. Here, worms share a common pseudo random permutation of the Internet address space. Infected systems start scanning at a fixed point in the permutation. If the worm detects an infected system, it simply picks a new random point in the permutation and begins scanning again. This prevents reinfection and imposes a measure of coordination on the worm. Until the Witty worm [12], these strategies have not appeared in the wild.

A.2 DNS Review

DNS is a globally distributed hierarchical database that provides a mapping between numeric IP addresses and alphanumeric domain names. Whenever access to service occurs that uses a domain name to locate a server, DNS is used. A domain name is a human friendly pseudonym for a systemâs IP address.

DNS queries are performed on behalf of the user by a resolver, an application installed on the user's local system to query the local DNS server whose location is specified during the system's network connection configuration. The resolver contacts the local DNS server with the domain name provided by an application. If the local DNS server doesn't know the IP address for the requested domain name, it queries external DNS servers to resolve the domain name. If the external DNS servers do not know the information for the domain name, they respond to the querying local DNS server with the address of an authoritative DNS server higher up the chain. A server is considered *authoritative* about a domain if it can respond to a query with certainty that the name exists.

If a system or user has a priori knowledge of the IP address of another system it needs to access, a DNS query can be avoided. However, the majority of accesses to remote systems are initiated by specifying the domain name in a client application. DNS resource records are the discrete data structures used to store information about the structure and content of the entire domain name space. There exists a variety of DNS resource records. The resource record of interest with respect to our detection approach is the *authoritative resource record* or *A record*, which maps a fully qualified domain name to an IP address. The mapping between domain names and numeric IP addresses can change over time as new services are added or as networks change. Each DNS record has an associated *Time to Live* (TTL) value, the number of seconds that the mapping will be guaranteed to be valid. The TTL dictates how long the resource record will be kept in the DNS server's cache. Caching resource records enables a DNS server to reduce the number of requests it needs to make to other name servers. Although the TTL value can be as low as a few seconds, in practice the default TTL is one day.

TTL values are associated with all DNS replies. A TTL value provides a mechanism to allow resource records to expire so that the information they contain can be updated periodically in case changes to the network topology are made.

B Whitelists

Tables 6 and 7 indicates whitelists used (see Sections 4.2 and 4.3).

Table 6: Lab Whitelist

#	Activity	Whitelist Entry
1	Secure Internet Access Protocol (IMAP)	192.168.1.33:993 - 192.168.200.50:993 TCP
2	Secure Internet Access Protocol (IMAP)	192.168.1.25:993 - 192.168.200.50:993 TCP
3	Network Time Protocol (NTP)	192.168.1.12:123 - 192.168.200.2:123 UDP
4	Network Time Protocol (NTP)	192.168.1.12:123 - 192.168.200.1:123 UDP
5	Secure Internet Access Protocol (IMAP)	192.168.1.5:993 TCP
6	File Transfer Protocol (FTP)	192.168.1.0/191:21 TCP
7	Secure Shell Protocol (SSH)	192.168.1.0/191:22 TCP

Table 7: IDN Whitelist

#	Activity	Whitelist Entry
1	Secure Internet Access Protocol (IMAP)	192.168.200.50:993 - 192.168.1.33:993 TCP
2	Secure Internet Access Protocol (IMAP)	192.168.200.50:993 - 192.168.1.25:993 TCP
3	Network Time Protocol (NTP)	192.168.200.2:123 - 192.168.1.12:123 UDP
4	Network Time Protocol (NTP)	192.168.200.1:123 - 192.168.1.12:123 UDP

C IDN Worm Alerts

Table 8 provides supplementary information for section 4.3.

Table 8: IDN Worm Alerts

Date	W32.Sasser	W32.Blaster	W32.HLLW.Gaobot.gen
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

D False Negatives

Table 9 contains probabilities, for various parameters, that a false negative will occur due to a single scan simultaneously occuring from each infected system targeting an entry in the connection candidate list (see Section 4.4).

Table 9: Probability of False Negatives due to Non-DNS Monitoring

DNS Records	10 Infected Systems	100 Infected Systems	200 Infected Systems	500 Infected Systems
500	1.1641^{-6}	5.821^{-6}	1.1641^{-5}	2.328^{-5}
1000	2.328^{-6}	1.164^{-5}	2.328^{-5}	4.656^{-5}
2000	4.656^{-6}	2.328^{-5}	4.657^{-5}	9.313^{-5}
5000	1.1641^{-5}	5.821^{-5}	1.1641^{-4}	2.328^{-4}
10000	2.328^{-5}	1.164^{-4}	2.328^{-4}	4.656^{-4}

E Network Cells

Figure 3 shows an enterprise network divided into three distinct cells.

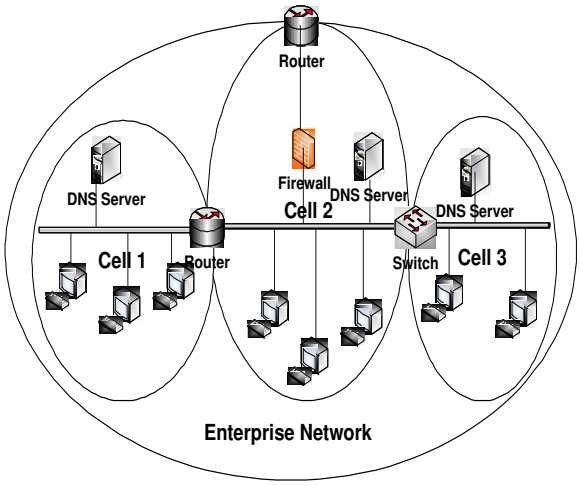

Figure 3: Network Cells

Figure 4 provides an example of how the prototype could be operationally deployed. Prototype A in Cell 1 monitors activity between Cell 1 and Cell 2. Cell 2 contains the sole ingress/egress point for the enterprise network. Prototype B, from its vantage point in Cell 2, monitors activity from all cells within the enterprise network to external systems. Finally prototype C monitors activity between Cell 3 and Cell 2. A system in Cell 1 is infected with a scanning worm. The infected system begins scanning to locate susceptible systems both within Cell 2 and the Internet. The prototype device in Cell 1 will detect the scanning activity to Cell 2 and generate an alert. The prototype device in Cell 2, at the enterprise gateway, will detect scanning activity from Cell 1 to the Internet and generate an alert.

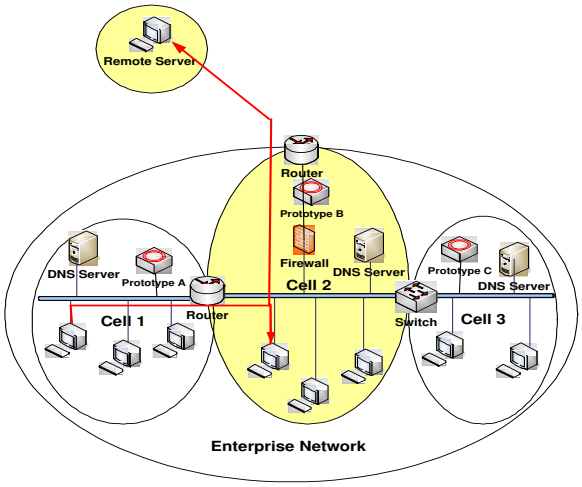

Figure 4: Worm Propagation Detection