Approved for Public Release; Distribution Unlimited
Case #06-0631

MTR-06W0000035

MITRE TECHNICAL REPORT

Graph-based Worm Detection On Operational Enterprise Networks

April 2006

Daniel R. Ellis
John G. Aiken
Adam M. McLeod
David R. Keppler

Paul G. Amman, George Mason University

Sponsor:	MITRE Corporation	**Contract No.:**	
Dept. No.:	G024	**Project No.:**	05MSR109
Derived By:		**Downgrade To:**	
		Declassify On:	

The views, opinions and/or findings contained in this report are those of The MITRE Corporation and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

©2006 The MITRE Corporation. All Rights Reserved.

MITRE

Corporate Headquarters
McLean, Virginia

MITRE Department Approval: ______________________________

Dale Johnson, Ph.D.

Department Head

Department G024

Project Leader Approval: ______________________________

Daniel R. Ellis

Principal Investigator

Automated Worm Detection & Response

MITRE-Sponsored Research Project

Abstract

The most significant open challenge to the worm defense community is to develop a sensitive detection method that can detect new worms in real time with a tolerable false alarm rate. This paper presents a graph-based detection system and validates it on operational enterprise network data. We argue that the result is significantly closer to solving this challenge than other published works.

We show that a graph-based approach to worm detection in an enterprise network can detect a broad range of active worms with a false alarm rate of less than twice per day. The supporting analysis comes from running the detection algorithm on a real enterprise network. The sensitivity results are significantly better than what is reported in the literature. We can detect all active, fast-spreading unimodal worms, including hit-list, topological, subnet-scanning, and meta-server worms.

1 Introduction

Worm attacks have been a fact of life for many years, and there is little reason to expect them to disappear. Although no one has yet released a worm with the saturation level and spreading speed that the academic community has shown to be within reach [10], modern worms have still managed to infect large portions of the Internet in minutes. For example, the Slammer worm [5] infected over 95% of all vulnerable hosts across the Internet in less than 10 minutes. Such exceptional speed was possible even when it used an ineffective targeting method (randomly selecting 32-bit IP addresses) and affected a small population (approx. 75,000 hosts). A targeted attack against these hosts could have infected them all in less than a second [10]. Naturally, the time from first infection in an enterprise network to saturation of that network is even less than the time required for saturation of the Internet. Further, worms may be launched without warning, and without foreknowledge of the exploits used. At these speeds it is impossible for humans to be an integral part of the detection and response process, thus we require automatic detection and response. Possible automated responses span a spectrum of impact to the enterprise network and include filtering traffic or quarantining of hosts identified as infected. All responses require a detection component to trigger and inform the response. The accuracy, sensitivity, and performance of the response system depends on the detection system.

To complicate the problem space, any response carries a considerable cost in terms of lost productivity, not to mention sheer annoyance on the part of network users, as responses *must* disrupt some network traffic–ideally, as little non-worm traffic as possible. In many cases, a worm-response system which routinely disables significant parts of an enterprise's network based on false alarms may be more damaging than simply enduring periodic worm damage. Thus worm-detection systems require very low false alarm rates (FAR), not only in terms of internal "alarm counts" within the detection system, but also in terms of the user-visible network impact the associated responses would have.

We developed a worm-detection system, based on detecting causal trees in graphs of communication behavior, and validated this system using a year-long trace of a major enterprise network. We believe that our results are significantly closer to solving some of the major problems in worm detection, as it is both sensitive (able to detect many classes of worms), accurate (a FAR which averages < 2 events/day), and efficient (able to run at super-real-time when processing traces).

Our worm-detection algorithm works by constructing a graph of network communication, and looking for specific behavioral signatures[1] which are indicative of a propagating worm. We then tune our algorithm's parameters to minimize the FAR, and use the resulting settings.

We validated our algorithm using several techniques. We evaluated both performance and accuracy by, first, tuning and then applying this algorithm to a separate nine months' worth of traces from a significant portion of a target network's operationally critical enterprise network. We collected complete header traces from several adjacent class-C networks, including a significant amount of intra-switch traffic.

Over the course of a year's worth of data, our algorithm produces an average FAR of slightly more than 3.1 per day, despite our test network's somewhat unstructured nature. We determined that there were three Domain Controllers (DCs) which were not on the monitored networks, but who's communication created a worm-like signature on port 137. We also detected a corporate network scanner sweeping the internal network for vulnerabilities. White-listing the DCs and scanner reduced the FAR to an average of 1.6/day. (Un)Fortunately for (us) the target network, there were no worms observed in the data.

Our prototype is also fast, able to process these traces at over 10x real time, despite an inefficient Java implementation. Additionally, since our algorithm is based on communication patterns, we do not need to perform deep packet inspection, reassembly, or other complex network transformations.

For sensitivity, we manually constructed graphs which represent typical infection scenarios for worm outbreaks, including hit-list worms using pessimistic assumptions. We show that these graphs are distinct from normal traffic that our algorithm can detect them quickly. In the case of a worm with the spreading properties of Zotob, we show that it is feasible to detect the presence of the worm on the target enterprise network when only one of the 600 vulnerable hosts on the network has been infected by the one originally infected host.

Paper Outline The paper is outlined as follows. Section 2 describes the requirements for a real-time worm detection system. Section 3 compares our approach with the literature and articulates our contributions. Section 4 provides the detection model. Section 5 describes the system deployed and its environment. It also contains a description of how the system was tuned and how one would expect to apply this system in another environment. Sections 6 and 7 provide an analysis of the sensitivity and accuracy of the approach, respectively, for a given set of thresholds. Section 8 summarizes our conclusions and presents future work.

2 Real-Time Worm Detection Requirements

Weaver et al. [14] articulated three key metrics for evaluating a worm detection system: accuracy, performance, and sensitivity. We demonstrate that our approach is more sensitive than what is described in the literature, with acceptable accuracy.

We also provide evidence that it can be stood up in real-time.

2.1 Accuracy

We use two metrics for accuracy. The most familiar is the FAR. Since multiple alerts related to the same activity may sound in quick succession, the FAR alone does not relate well to the operational experience of a typical intrusion detection analyst (IDA). If a series of alerts related to the same alarm are generated in quick succession, the IDA will normally only count them as one "alarm."

This gives rise to the notion of an alarm *deadband*. Each alarm corresponds to a temporal cluster of one or more alerts related to the same event. The resulting deadbanded FAR is normally lower than the undeadbanded FAR and reflects an IDA's operational concept of a FAR. The FAR of 1.6 per day we claim for our system on the target network and use throughout the paper is the deadbanded FAR.

Deadbanding alarms has not yet been applied in intrusion-detection-related work, although its conceptual cousin–correlation–has been an area of active research. This is because the signal being deadbanded (alerts) are of such different types and relates to different events. However, in our system they are all of the same type and it is often the case that one alert (indicating a single host to be infected) will be quickly followed with others. Measuring a detection system this way more closely approximates impact on the IDA's experience. They are going to care more about how often the automated worm detection system needs looking at than the number of diagnosed infections associated with an alarm. In essence, the alarms raised by our system are inherently correlated. Some may see deadbanding as a trick to lower one's FAR. To counter-balance that concern, we complement the FAR using the typical signal processing technique of marking the duration of the deadband (the time window for which we keep state) as being in an alarm state for its entire duration. We count not only the number of false alarms but the amount of time that the "alarm light" is on, which leads to our next accuracy metric.

The *annoyance percentile* (AP) reflects the amount of time the alarm "light" is on[1]. Our approach achieves an AP of less than 2%. This implies that the alarm light is on less that 30 minutes per day. Using both metrics to measure the accuracy of an automated worm detection system accounts for the impact of the false alarm rate on the operational IDA in a a novel and more meaningful way than simply using the FAR.

The approach described in this paper has been validated on roughly a year's worth of operational, internal enterprise network traffic. The target enterprise has

[1]Just because the detection system's light is on does not mean the corresponding response system is active. These accuracy metrics approximate the negative impact of FAR rate on the operational IDA, not on the end-user.

an academic culture: there are few constraints on the choices in applications and approaches employees use to achieve results. With such an unconstrained environment we expected to see lots of unexpected behavior, some of it worm-like in its manifestation. While unexpected traffic patterns were observed, only a small percentage was worm-like. There are network applications that create topologies that have worm-like features and they are discussed at the end of the following subsection. We do not claim to detect worm that target P2P applications.

2.2 Sensitivity

Sensitivity measures how many true alarms are actually detected. It is impossible to measure sensitivity perfectly without either a general model for all worms of interest, including their effect on the normal network traffic, or an emulation of all worms of interest. Since we have neither of these, we are limited to analytical scenarios. The data collected so far is ideally suited to accuracy analysis, but not to sensitivity analysis. We are working on a software tool that will safely generate configurable, emulated worm traffic across operational network environments.

An approach to the worm detection problem can be evaluated by the subspace of worms which can be detected. Most published detection approaches to date are able to detect subclasses of worms based on the spreading strategy and polymorphic attributes. In addition to the qualitative subspace evaluation, we can evaluate worm detection systems' sensitivity using historical or well-defined hypothetical worms. We propose two analytically computed performance metrics, 1) *time to detection*, and 2) *number of infected hosts at time of detection* to evaluate the sensitivity of a worm detection system. Time to detection can be measured either in terms of seconds or number of scans. The number of infected hosts can be measured either in terms of percentage or raw number of vulnerable hosts infected by the specific pathogen when detection occurs. Given the same worm, network, and initial infection point, a quantitative comparison can be made between two differing detection systems for that test. These "analytic performance metrics" are not true performance metrics because their computation does not take into account the computational delay which may be imposed by a finite amount of computing power. However, these metrics adequately describe the sensitivity of the system's underlying detection model.

Our approach can detect all classes for which a detection claim stands in the literature and more classes still. The sensitivity of our approach is illustrated by evaluating these "analytic performance metrics" for the Zotob worm and for hypothetical worms that transform the Zotob case into worst-case scenarios.

If the Zotob worm had appeared on the target network in a scenario with worst-case assumptions, the expected value of the time to detection would be 73 scans

(1.8 seconds) while minimum and maximum values would be 22 and 92 scans, which correspond to 0.6 and 2.3 seconds, respectively. The expected number of infected hosts at the time of detection would be 1.8 (including the original infection) and this is 0.3% of the 600 hosts in the vulnerable population. The minimum number of infected hosts at detection time would be one (namely, the initially infected host). This occurs when no new hosts have been infected by the worm and represents 0.16% of the vulnerable hosts. The probability that no new hosts would be infected before detection time is about 0.4. The theoretical maximum number of infected hosts at detection time would be 106 (about 17% of vulnerable get infected), but, the worm's random number generator has to mimic a hit-list worm for this case to occur. These numbers all include worst-case assumptions. These assumptions are stated and discussed at the beginning of Section 6.

We claim that our approach is sensitive to all classes of active[2], fast-spreading unimodal worms. It has been observed that operational network traffic *could be crafted* to look worm-like. For example, it is conceivable to create a networked business application that has communication patterns or topologies that, from certain perspectives, look like a worm (e.g., OpenDHT and P2P file-sharing applications that do flooding[3]). Although our approach could be used to detect such spread, it would also detect lots of normal (P2P-like) behavior and would therefore need to be desensitized (in order to achieve an acceptable FAR) possibly to the point where detection happens after considerable spread has already taken place. In the data we gathered, we have not found any evidence of such topologies giving us difficulty. Many network applications installed on our networks *could be* used to create such topologies. For example, NFS, web services, and file sharing applications (e.g., gnutella) run on the target network, they have not behaved worm-like, at least, not with respect to the features we have chosen to measure. We consider this both comforting and unfortunate. It is comforting to know that such topologies are rare (at least in our environment) and that worm detection is feasible. It is unfortunate from a technical perspective, because we cannot provide experimental data outlining the boundaries of our approach. Nevertheless, our claim stands that our approach can detect all active classes of worms, and, in our environment *will* detect them. Whether that claim stands for other environments will require an accuracy and sensitivity analysis using data from their network.

[2]An active worm initiates network traffic in order to spread.

[3]A worm that spreads through a P2P infrastructure can spread in a way that makes its behavior indistinguishable (from layer 4 and below) from normal P2P behavior. Worm detection in a P2P environment is an open problem.

2.3 Performance

A perfectly accurate and perfectly sensitive system is not useful unless it can produce the result in the time frame of the worm attack. Naturally, assuming no degradation in sensitivity or accuracy, faster is better. This metric is not useful alone, but is of significant importance when the detection system is coupled with the response system. We save a thorough analysis of the performance aspects of our approach for future work, although we do make the case that it appears feasible to do in real time. In this paper, when temporal issues are addressed they are addressed at the model level and address the question "when is there enough data to confidently sound an alarm?".

Our algorithm runs at least an order of magnitude faster than wall clock time for the packet traces collected. The most significant bursts have resulted in the system running at roughly 50% wall clock times (and occur approximately once weekly). The system is implemented in Java and was designed to evaluate the sensitivity and accuracy of the approach, with performance not being a primary design goal. That the implementation performs so well without performance being a design goal is encouraging news.

3 Related Work

The space of all possible worms can be meaningfully partitioned into subspaces. The defenses proposed are typically designed to address a subspace. It is possible to compare approaches qualitatively by comparing the subspaces covered by the respective approaches. To compare this paper to related work in the literature, the following division of the worm space seems appropriate.

- **Modality (unimodal, multimodal)**: A unimodal worm uses a single exploit in every attempt to spread. A multimodal worm may spread using more than one exploit. The network footprint (pattern of network emissions) of multimodal worm propagation may be much more complex than that of unimodal worms.
- **Polymorphism (yes, no)**: A polymorphic worm varies its network footprint on successive attempts to spread. It may still be unimodal, but its exploit may be expressible in a variety of ways.
- **Target Acquisition Function (TAF) (random, subnet, hit-list, topological, meta-server [12])**: A worm has a variety of methods of choosing its next target. The simplest are random scanning worms, which randomly attempt to infect hosts in the 32-bit IP space. Subnet scanning, such as employed by Zotob, limits scanning to a subnet (e.g., using a the /16 network prefix of the infected host). Hit-list worms attack predetermined lists of hosts. Topological worms

use information local to infected host (e.g., `.rhosts` files) to identify targets. Meta-server worms use external data compilation services to identify targets (e.g., Google can be used to gain information on vulnerable web servers [13]). Undoubtedly, other TAFs are possible, but, to our knowledge, there are no such TAFs described in the literature.

- **Frequency Threshold (Hz scale)**: Worms with low attack rates are, other things being equal, more difficult to detect than worms with high frequency attack rates.

In addition to worm space divisions, the following dimensions can be measured for qualitative or quantitative comparisons between worm detection approaches.

- **Data Source for Validation (purely theoretical, emulation, telescope, network perimeter, enterprise network data)**: Validation of approaches requires some data source. In increasing order of realism, sources reported in the worm literature fit into the four categories listed above. A purely theoretical evaluation derives detection conditions, but typically does not consider complicating factors unavoidable present in real networks. Emulating (manually constructing and emitting) benign or malicious network traffic provides only proof-of-concept validation. Although network telescopes, which generally observe (misdirected) network traffic, provide "real" data, they do not provide network traffic representative of a functioning enterprise environment. Perimeter data contains data visible by telescopes and some portion of inter-enterprise traffic. This data is useful for detecting some types of worms or protecting Internet-facing servers, but does not resemble traffic within the enterprise. Enterprise (internal) network data is, by definition, realistic, although still somewhat idiosyncratic to the particular enterprise being monitored.
- **Accuracy (FAR and AP)**: A worm detection system's accuracy can be measured quantitatively given an operating environment. The FAR and AP fix the user-visible experience.
- **Sensitivity (time to detection)**: A detection system's sensitivity can be measured quantitatively for specific worms. We propose using *time to detection* or *number of hosts infected at detection time* as useful metrics sensitivity. This dimension fixes the utility of the system in detecting worms.

Williamson [15] was the first to propose and implement an approach to worm detection and response. The approach measured the frequency (measured in unique destination IP addresses per unit time) to drive detection. The response was to throttle and then eventually contain infected hosts (i.e., blocking their network access altogether). This approach is effective against worms that try to initiate more connections per unit time than his threshold allowed (5 Hz). In terms of the worm space above, this approach is effective against all of the subspaces identified above, but at the cost of an unacceptable FAR for active hosts on a network. A more detailed analysis of worm behavior, such as is present in subsequent works, including the present work, is needed to bring the FAR to within a reasonable level. Williamson's approach was validated using emulated data that had worm-like traffic but no "normal" traffic, so a published FAR on enterprise data is not available for comparison.

Staniford [9] advanced this approach and showed that he could effectively contain a scanning worm within 10 egress flows. This approach is not sensitive to the frequency of unique flows per second, but is sensitive to the frequency of unique unsuccessful flows, which are expected in very high quantity when a worm selects random scanning TAF. Other improvements [7] enhance performance but do not increase sensitivity to worms outside the class of random scanning worms.

In terms of the classification above, these works cover unimodal worms and have little tolerance for polymorphism. These approaches only claim sensitive detection to random scanning worms (although the claim may be extendable to cover subnet scanning worms for certain parameters of their approaches). By spreading with a more sophisticated targeting algorithm (e.g., a hit-list or topological TAF), a worm can evade Staniford's more comprehensive approach. However, within the class of random scanning worms, Staniford's approach is a very sensitive system with acceptable accuracy. With respect to frequency of activity, these approaches can detect worms that spread with a very high branching factor per unit time. However, by spreading at below 5 Hz, a worm can evade both of these approaches.

A decade ago, Staniford et al. [11] implemented GrIDS following a graphical approach to detecting hackers as they move across a network in a "leap-frog" manner. They also observed that worms and other mobile malicious code can spread across the network in anomalous ways. However, it is not clear what features GrIDS extracted. Unfortunately, GrIDS has not been supported for many years, and so is not available for direct comparison on our network data. At the time, worms were poorly understood and no sensitivity or accuracy analysis was performed with which to compare.

Another approach is to look for common byte sequences in disparate packets across the network. When a common byte sequence of sufficient length propagates across enough network flows within a sufficiently small amount of time, it is deemed to be a worm. This byte sequence can be extracted and distributed as a signature for a specific worm. Two implementations of this approach are Autograph [3] and EarlyBird [8]. Polygraph [6] relaxes the constraints of EarlyBird and Autograph so that smaller and disjoint string patterns can be used. In terms of the classifications above, these approaches make no claims of being sensitive to anything beyond unimodal, polymorphic worms with random scanning TAFs. Since signature-based approaches identify misuse directly, they work at any frequency of attack. EarlyBird used university campus network traffic of some type. Polygraph was validated with fifteen days of network perimeter traffic. Using the dimensions above, our contribution over the state of the art are as follows:

Modality: We detect all unimodal and some multimodal worms. The multimodal worms we cannot detect (called discriminating worms in [1]) are those where the same exploit is used only once in a given ancestry chain (more accurately, the portion of the ancestry chain visible in the observable network traffic). With the exception of Williamson, the extension to multimodal space is novel in this paper.

Polymorphism: Since our approach is payload agnostic, we can handle all polymorphic worms. This offers an extension over Newsome et al. in that we do not depend on their being invariants within the exploits that a worm might use.

TAF: We can handle each of the listed TAFs. Our work is an advancement over the literature in that the "workable" systems cited above are generally limited to random scanning worms, possibly extending to subnet scanning worms.

Frequency Threshold: The frequency is determined by the network observation window, a tunable parameter in the system. For this paper data was analyzed with a feature window size (FWS) of 300 seconds. Any worm that breaches a feature within this window can be detected. Worms with a spread rate on the order of 10^{-2}Hz can be detected[4]. Adapting this to cover slower spreading worms can be done by increasing the FWS and linearly increasing the memory footprint. With respect to systems that look for worm activity above a given frequency, our system monitors at a very low frequency—certainly low enough that human intervention is possible for worms that spread below that frequency. Contemporary signature-based systems generally have no frequency constraints at the model level, and so have an advantage over our work in this respect.

Data Source for Validation: Our system was validated with operational enterprise network traffic (see Section 5 for more details). This level of validation is more extensive than that reported in any other paper in the literature.

Accuracy: Our approach's accuracy is comparable with what is reported in the literature. Further, we provide novel accuracy metrics, which, we believe,

[4]In order to spread slowly enough to evade detection, a worm cannot have more than a few descendants (for most ports) in a 300-second period of time.

are more meaningful.

Sensitivity: We claim sensitivity that extends beyond what is reported in the literature. We also propose and use a novel metric (time to detection) to report our system's sensitivity against real and hypothetical worms. Staniford's approach to worm detection would have a slightly smaller time to detection than ours, given our current thresholds, for random scanning worms. We willingly trade off a slight degree of sensitivity to random scanning worms to get the added sensitivity to other classes of worms while preserving acceptable accuracy metrics. We could re-tune for increased sensitivity to random scanning worms if desired. With the thresholds Staniford and Williamson both published, we observed an unacceptably high number of false alarms on the data collected.

Xie et al. [16] also uses the observation that the spread of worms is tree-like and causal. They use this observation, however, for forensic analysis and to provide traceback, not to drive detection.

To summarize our contributions, we apply a novel worm detection technique and associated features to operational enterprise network traffic. We propose new metrics for evaluating worm detection accuracy (deadbanded false alarm rate and annoyance percentile) and sensitivity (time to detection) and show that our sensitivity extends beyond the literature, while maintaining acceptable accuracy. There are three caveats to our claims about detecting all unimodal worms. First, we address only active worms (i.e., not viruses, email-borne viruses, or contagion worms). Second, we do not address worms that spread across P2P applications. And, third, a worm can spread slowly enough to evade detection.

4 Model

We use a model comparable to that in [1], which describes networks and network communications and shows how to represent them within a network. Worms appear as aberrations within this model. Primary amongst the manifestations is the causal, tree-like spread that is inherent in worm spread but rare otherwise.

4.1 Network Model

Our network model is a graphical model that represents end hosts as nodes and network traffic as edges. Link predicates are evaluated within the context of two hosts and the traffic between them. A packet is modeled as a link, (a, b), between two hosts a and b. A packet trace, L, is modeled as a temporally ordered sequence of links. A link predicate describes the traffic that must flow between two nodes for it to be true and can require one or more packets to flow between a and b. A link predicate is defined by the following.

Definition A *link predicate*, $P_{L,t}(a, b)$, is true with respect to two hosts, a and b, at time t if there is network traffic between a and b that has completely satisfied P at time t in the packet trace L. L and t are omitted when context allows. P can specify traffic flow in either direction. The following are sample link predicates. $P_t(a, b)$ is true iff at or before t:

1. a sends b a TCP:80[5] SYN packet and then b replies with a TCP:80 SYN ACK packet
2. b sends a a UDP:69 packet
3. a sends b an ICMP Echo Request, then b sends a an ICMP Echo Reply, then a establishes a TCP connection (three-way handshake) with b on port 445, then a sends b a TCP:445 FIN packet

Link predicates can be arbitrarily complex. They can describe behaviors that flow in either direction; that is, $P(a, b)$ can be specified to be true if b sends a or vice versa. The temporal ordering of packets is important. For example, if a initiates a TCP:445 connection with b and then sends an ICMP Echo Request packet to which b replies, this would not satisfy link predicate 3, above. Link predicates can be chained together to create a *descendant tree*, which represents the causal flow of a particular behavior emanating from a host across the network. In our model, the graph emanating from the root is indeed a tree because we prevent descendants from being descended more than once.

The *descendant set*, D_a, is the set of all descendants of the root, a. We preserve the depth of each descendant in the tree with an added superscript: D_a^i is the subset of a's descendants that are at depth i in the descendant tree. The time, depth, and packet trace are omitted when they are clear from context or are insignificant. For example, we can write $D_a = \bigcup_i D_a^i$.

In summary, the descendant tree is genealogical in nature and captures the causal spread of a particular type of behavior evidenced between hosts across a network. The link predicate defines the relationship between one generation and the next. As a worm spreads, the descendant tree preserves the parent-child relationships as the worm's descendant tree grows in depth and breadth.

[5]TCP:80 is a TCP packet sent to port 80.

4.2 Worm Propagation Model

Worms are limited in the number of ways that they know how to exploit and infect other hosts. They have well defined algorithms that specify the exact sequence of network communications that are necessary in order to infect a target host. This often includes 1) reconnaissance, 2) sending an exploit, 3) getting the worm's code on to the target host, and 4) commands to activate the worm on the target host. Different worms implement these steps in different ways, often omitting or combining steps. Regardless, once a worm is written, its necessary network behaviors become fixed. A worm may interject additional traffic to the target or other hosts, but the traffic necessary for infection is constrained by the port, protocol, vulnerable application, and possibly other parameters defined by the configuration of the target system. Each of these network behaviors can be described using a link predicate. Further, an aggregate link predicate can be written that describes the entire sequence of network behaviors necessary for infection. For example, the following describe the link predicates of historical worms.

- $P_{Welchia}(a, b) =$
  - a sends an echo request to b
  - b sends an echo reply to a
  - a connects to TCP port 135 on b and
  - b sends a UDP packet to a port between 666 and 765 on a
  - a replies with UDP packet with worm code
- $P_{Sasser}(a, b) =$
  - a establishes a TCP connection with port 445 on b and
  - a establishes a TCP connection with port 9996 on b and instructs it to download the worm code
  - b establishes a connection with port 5554 on a and downloads the worm code
- $P_{Zotob}(a, b) =$
  - a establishes a TCP connection with port 445 on b and
  - a establishes a TCP connection with port 8888 on b and instructs it to download the worm code
  - b establishes a connection with port 33333 on a and downloads the worm code

4.3 Features and Link Predicates

A link predicate can be crafted for any sequence of necessary network behaviors. Some link predicates may be sensitive to more than one worm. For example, there are several variants of most historical worms for whom the necessary network behaviors are similar. Further, even entirely different worms that exploit different vulnerabilities, may have link predicates in common (e.g., Sasser and Zotob can both be described with the same link predicate that captures a TCP:445 connection). The more fully a link predicate captures the necessary network behaviors, the more accurate it becomes.

As a worm spreads, the resulting descendant tree has some interesting properties. For example, the number of descendants at each depth grows relatively quickly. The features presented (though not thoroughly discussed due to space constraints) in this paper are:

- Sum the total number of descendants
- $Deepest$ the depth of the deepest descendant
- $AvgBpL$ the average branching factor per level
- $AvgBpP$ the average branching factor per parent
- $SumDlogB$ the depth-weighted sum of all branching factors of children that are also parents (scaled down logarithmically)
- $SumbBlogD$ the branching-factor-weighted sum of all depths of children that are also parents (scaled down logarithmically)
- $SumBD$ the sum of all branching factor by depth products of children that are also parents
- $Sum_{i=1..10}$ the total number of descendants at depth i
- $TTD_{i=1..10}$ the time it took for the root to have its first descendant at depth i

The link predicates we have chosen to present in this paper are fairly general, each detecting a large class of worms. We show the link predicates as composites of each other to demonstrate their relationships and make the description as succinct as possible.

1. $L_{UDP:Port}(a, b)$: a sends b a UDP packet
2. $L_{SYN:Port}(a, b)$: a sends b a TCP SYN packet
3. $L_{SYNACK:Port}(a, b)$: $L_{SYN}(a, b)$, then b sends a a TCP SYN ACK packet
4. $L_{TWH:Port}(a, b)$: $L_{SYNACK}(a, b)$, then a sends b an ACK packet
5. $L_{FIN:Port}(a, b)$: $L_{TWH}(a, b)$, then a sends b a TCP FIN packet

The $Port$ annotation in the link predicates above indicates that the link predicates are port specific. That is, $L_{SYNACK:445}(a, b)$ will only be true if a sends b

a TCP SYN packet to port 445 and b sends a a TCP SYN ACK packet back from port 445.

Each link predicate-port pair will have its own descendant relationship computed, and, all of the features described above will be computed for each descendant relationship. For each host observed within an area of interest all of the descendant relations will be computed along with all the features of those descendant relationships. For example, $LP_{SYN:445}Sum_2$ is the sum of the descendants at depth two where the parent host only needs to send a SYN packet to port 445 on the target host for a link to be defined. We provide the data for the first two link predicates, *UDP* and *TCP SYN*, in the appendix. Together, these two link predicates are sensitive to all unimodal worms. An alarm is sounded when any feature of any of the descendant relations breaches a threshold for any host. How we computed the vector of features to use (the "Operating Point") is discussed in later sections.

5 The System

In this section, we describe the system we have deployed, show how it reflects the detection model presented previously, and how we tuned it. We describe 1) the network environment we monitored, 2) how we monitored the network, 3) the detection algorithm, 4) the data structures we use, 5) how an operating point (OP) was obtained, 6) how the system was tuned on the target network, and 7) how the system could be tuned in other environments.

The Network Environment The data gathered for this paper represents packet traces coming from one building, which includes approximately 400 technical, support, and corporate staffers. The hosts are predominately Windows, but several other operating systems are in non-trivial numbers. The hosts are either workstations, production servers (two server farms), or experimental/laboratory equipment (one lab of a couple dozen machines). The number of hosts online at any moment in time ranges from 300 to 600. The building spans seven class-C-sized subnets.

The Sensor Only one dual-homed sensor was needed to cover this building. It's a Dell PowerEdge 650 with a modest x86 processor and 1 GB of RAM.

The Data The packet traces contain the first 68 bytes of each packet that crosses the network in this building. We have run the analysis on most of the traffic covering the year. The values of features of interest to us on that data were fairly uniform throughout the year. In this paper, quantitative data is only presented for one month from the year.

Analysis Engine The analysis engine currently runs in off-line mode over data previously gathered. This section outlines the data structures used and the algorithm used to compute the features described previously.

The analysis engine has the following configurable parameters:

1. the feature window size (FWS, the length of time over which features in a host record are computed)
2. the network address range of interest (the target building's IP address range)
3. the link predicates to use in building the descendant data structures (we present data on the first two)
4. the list of ports to monitor (data for ports 22, 25, 80, 135, 137, 138, 139, 194, 443, 445, 1434, 2049, 6346, 6669, 6699, 6881, 7000, and 9911 are presented)
5. the list of white-listed IP addresses (DCs)

Data Structures A $hostrecord$ is kept for each IP address observed in the packet trace that comes from the network area of interest. Each $hostrecord$ maintains a separate $descendant$ structure for each $linkpredicate$ provided in the configuration file to the system. The $descendant$ structure is a tree, corresponding to the descendant tree rooted at the host represented by this $hostrecord$. The features that drive detection are calculated over each of these trees. (Sum_i and TTD_i are evaluated out to depth seven.)

Algorithm The high-level algorithm for the detection engine is as follows. Its computational complexity is $O(P \times H \times LP)$, where P is the number of packets, H is the number of active hosts located on the monitored part of the network, and LP is the number of link predicates being checked. The memory footprint has the same upper bound, but is practically $O(H \times LP)$, because P's impact is practically constant. The current version of this algorithm runs only in off-line mode.

```
for each P: Packets
for each H: Active Hosts on this part of the network
  for each LP: Link Predicates to compute
    H.LP.update( P )  //O(1) computation
```

Obtaining an OP An OP is a set of thresholds and FWS for operating the detection algorithm. The OP presented in the appendix was derived from one month's data and its effectiveness is discussed in the following sections. The ports selected for presentation were the most actively used in the data analyzed and reflect different application types.

In the off-line version, the system outputs the features of each host seen in each feature window. The values of the features are those computed at the end of the respective feature window. Using the off-line output, we searched the multidimensional space for thresholds that would, when run together, combine to sustain an optimally sensitive system that satisfied our accuracy constraints: a FAR of less than twice per day and an AP of less than 30 minutes per day.

An observation about the values of thresholds aids in simplifying the tuning of the search algorithm. When all values for a particular feature are sorted in decreasingly sensitive order (one value per host record)[6], the associated curve tends to have a very steep drop off relatively close to the most accurate possible value[7]. What this means is that worm-like behavior is very rarely engaged in by normal hosts under normal operating conditions. By looking at the curve and selecting a point on the y-axis (the threshold value), the x-axis tells the number of host records that would have breached that threshold and resulted in a false alarm[8].

The implications of this property of the data are interesting. One is that there is always a threshold that is maximally accurate—one value above the greatest value seen. We have confidence that such a value occurs infrequently (i.e., less than once a month). Also, with a modest decrease in accuracy sensitivity can be increased significantly as one chooses a point on the curve further to the right. This proportional increase in sensitivity continues throughout the drop off. It is interesting that each of the features we chose tends to have a similar steep drop off relatively close to the most accurate thresholds possible. What this means is that the features are good discriminators between normal behavior and worm-like behavior.

The greedy algorithm we used to search for an OP is as follows. Choose the most accurate value for each threshold to start with. Then, iteratively sensitize each threshold while the accuracy constraints are satisfied. It is conceivable that there is a more favorable OP than the one derived.

Figures 1, 2, and 3 are histograms that illustrate the busiest features we observed in choosing an OP[9]. TCP:445 was the nosiest port, and therefore required the least sensitive thresholds. Figures 1 and 2 are histograms of sorted (in descending order) measurements. These graphs demonstrate that, even for this noisy port, sensitive thresholds can be found with respectable accuracy. Figure 1 illustrates that a Sum threshold of about 100 and a Sum_1 threshold of about 92 would have no false alarms. Figure 1 also illustrates the value for graphical (tree-based) approaches: even for the noisiest port, there is still a general decrease in the number

[6]The Time-to-Depth (TTD_i) thresholds are breached by having values less than the threshold and represent causal spread faster than the norm. All others are breached by having values greater than the threshold. For simplicity in speech, when we speak of "increasing" a threshold, we mean decreasing its sensitivity, even though it is backwards for the TTD_i thresholds.

[7]Note that the graphs show only the largest 10,000 values. For a single month, several million host records are created.

[8]Note that the x-axis is truncated at 10,000 records, although for a month-long run there were many millions of host records.

[9]All graphs are from the baseline data, which is assumed to be free of worms.

of descendants at greater depths, which is exactly opposite of worm spread. For other ports, the drop off is steeper and closer to the y-axis.

Figure 2 illustrates that the relative number and progeny of descendants that are also parents are small values. BD, which is calculated by summing all the $branchingfactor \times depth$ products of children that are also parents, dominates because there are a few very chatty descendants that have a branching factor of about 30 at depth 5.

While tracking down the hosts that had caused false alarms, we noticed that the large majority of alarms were caused by three hosts—all were DCs. Once the three DCs were put on the white list, there were very few descendants at any particular depth. The effectiveness of these chosen thresholds (including the feature window size, FWS) is discussed in the following sections on sensitivity and accuracy.

The possibility of the white-list causing significant harm is small. The worm will still be detected if it spreads from the white-listed hosts to others and thence to others within the enterprise, or from other hosts to the DCs. However, to protect these few servers (whose value is significant), we recommend complimenting the approach presented with another (e.g., host-based) approach. The gain in sensitivity achieved by white-listing the domain controllers, however, is significant.

The final value to be tuned is the feature window size (FWS), which specifies how long a $hostrecord$ is preserved in memory and over which time all features are calculated. There are three issues that affect the FWS. First, to combat our system a worm may spread so slowly that not enough of the descendant tree is ever in memory at one point in time. The FWS must be long enough to force a worm that uses such a strategy to spread slowly enough for a human-based response time be possible (of course, the human could take advantage of similar or other tools). Second, the FWS cannot be too long, otherwise too much state is preserved and memory consumed. (This factor did not ultimately affect our decision.) Third, the accuracy metrics which we use to evaluate the approach requires that the FWS be the smallest value that the alarm light can go on. An increase in the FWS that does not impact sensitivity only negatively impacts the theoretical measure of the system's accuracy (i.e., AP), because the alarm light stays on longer.

To inform our decision on what FWS to use, we evaluated the FAR and AP of FWS of sizes between 30 and 1200 seconds. In order to satisfy the accuracy constraints (specifically the AP constraint), the larger FWS runs required less a less sensitive OP (determined using the greedy algorithm as described above). We chose a 300-second FWS as an OP that satisfied our operational expectations. The figures and appendices are all for a 300-second FWS. Note that a 300-second FWS does not mean that a human has to respond within five minutes, it just means that is how long state is preserved. Practically speaking, a worm must not have more than two grandchildren in a five-minute span of time. The implication is that worms must spread fairly slowly to avoid being detected with a 300-second FWS. In such a case, a purely autonomous system is not necessary—a human engage in the detection and response process.

The thresholds chosen for the the link predicates and ports previously described is provided in the appendix. For the (TTD_i) features, a threshold is breached and an alarm is sounded when the value of the feature is less than or equal to the value of the threshold. For all other features, the threshold is breached when the value of the feature meets or exceeds the threshold value. (For example, a value of 91 or greater breaches the $2[445] : Sum_1$ threshold and a value of 6352 or less breaches the $2[445] : TTD_3$ threshold.)

For most ports, having two descendants at depth two (i.e., grandchildren) or having any grandchild within a minute of observing the root is enough to sound an alarm. For these ports, it would be possible to consolidate to a single vector of 21 thresholds. The few exceptions are the busiest monitored ports: UDP:137, TCP:139, and TCP:445.

A worm that continues to spread has a descendant tree that grows in depth and breadth. Eventually, the number of descendants at a particular level will sound an alarm. Also, a worm, if it spreads quickly enough, will also breach a TTD threshold, even if it does not spread across a large number of hosts.

A worm which attempts to defeat our algorithm must either 1) avoid detection by spreading so slowly that the features are never breached within a feature window, or 2) assume that it is going to be detected and try to maximize the number of machines which it can infect. A class of fast-spreading worms which attempt to maximize the number of machines they infect before they are detected is presented in Section 6.

To summarize, the tuning process we went through with the detection system consisted of choosing accuracy (i.e., FAR and AP) constraints. We then white-listed the few badly behaved hosts. The OP chosen by this process was untouched and used directly in the accuracy and sensitivity analyses that follow[10]. It is not clear whether the OP we have selected would be as sensitivity and accurate on a different network. We expect that it will be close, because the underlying client-server model, according to which most network applications are designed, is the dominant model. Likewise, it requires more study to determine how the accuracy of the system scales over the size of the network being monitored (e.g., up to a 10,000-node network). However, applying the same model here leads us to expect that descendant trees will remain fairly shallow—there will just be more of them. We are actively seeking other networks to test the system on. In any case, with a packet trace, tuning is straight forward.

[10]The OP was selected *before* Zotob was released.

6 Sensitivity Analysis

In this section we describe the sensitivity of our approach by taking a recent worm example (the Zotob worm) and showing when our model would have detected the worm in an operational setting. We also show that we could detect Zotob, even if the author had made it to be a hit-list worm instead of a subnet-scanning worm. A key point to make is that we used no *a priori* data on Zotob before doing this analysis. The thresholds used were developed before Zotob was released.

This section highlights our most significant contribution: the ability to detect previously unseen worms of all types of TAFs. The TAF-independent sensitivity is a result of focusing on the features inherent in spread, which are captured in the descendant tree structure, and not on superficial expressions of worm payloads or targeting strategies.

6.1 Zotob Worm Description

The relevant features of Zotob are as follows [2]. The Zotob worm spreads from one enterprise to another when an infected host is physically moved from one enterprise to another. The Zotob worm exploits a vulnerability in the Windows 2000 operating system. Hosts running other operating systems are not vulnerable. The Zotob worm sends out its exploits on port 445. A host infected with the Zotob worm sends an exploit to a randomly chosen IP address in its own /16 address space about once every 25 milliseconds. To infect a host, the Zotob worm 1) sends out its exploits on port 445 and then 2) opens a connection over one of 11 ports in the 8xxx range and then 3) opens a connection on port 69. All this must happen before an infected host starts sending out exploits of its own. On a typical network, the delay is on the order of a 100 ms before an infected host becomes active.

6.2 Example 1: 600 Vulnerable Hosts

In this example, 600 of the target network's 8k hosts are vulnerable and are in the Zotob worm's 64k IP address space.

Suppose that a host infected with the Zotob worm was connected to the target network when 600 or so Windows 2000 hosts on the target network were vulnerable. In this section, we show that our algorithm is expected to detect this hypothetical infection in 1.8 seconds after the worm first appears on the network and that expected number of infected hosts at the time the worm is detected is 1.9. This number (1.9) includes the initially infected host. These expected values (1.8 seconds and 1.9 hosts) are estimates of the average detection time and average number of infected hosts where the average is taken over all possible seeds for the worm's pseudo-random number generator. These estimates are derived with "worst case" assumptions; they represent an upper bound to what would be expected in an operational environment.

The estimate of 1.8 seconds for the average detection time does not include the latency introduced by packet processing. Assuming that the initially infected host sends out its first exploit at time $t = 0$, the *detection time* or *time to detection* of 1.8 seconds is more accurately expressed as *the time when the worm has generated enough packets on the network to allow the algorithm to decide to sound an alarm*. The minimum and maximum values of the time to detection and the number of infected hosts at the time of detection are as follows.

1. The minimum reasonable detection time is 0.6 s.; this value occurs with probability 0.04. [11]
2. No new hosts are infected before detection with probability 0.4.
3. The max. detection time is 2.3 s.; this value occurs with probability 0.5.
4. The max. number of new hosts infected during the detection time is 105.

We make the following "worst case" assumptions.

1. All 600 vulnerable hosts are in the /16 address space of the infected host.
2. All 600 vulnerable hosts were on-line on the target network on 8/11/2005.
3. Each new infection starts its random number generator with a randomly chosen seed.
4. The worm is equally likely to choose any of the addresses in its /16 address space for its next exploit.
5. The packets generated by the initially infected host before time $t = 0$ have not satisfied any of the predicates for any of the computed features.

The first four assumptions imply that, in the initial phase of the infection, each of the exploits will successfully infect another host with probability 0.01 ($0.01 = 600/64k$). This in turn implies that each such exploit will fail with probability 0.99. If the initially infected host makes 92 scans, then the worm will be detected by the sum at depth one feature with the SYN predicate, $LP_{SYN:445}Sum_1$, unless it has already been detected by the sum at depth two feature with the SYN predicate, $LP_{SYN:445}Sum_2$. "One" is the minimum possible number of infected hosts at

[11] A 0.6 sec. detection time occurs when the first new host to be infected (the "second" infected host) is infected during the first four scans of the first infected host. Smaller detection times than 0.6 s. occur with negligible probability.

detection time. This occurs if no new hosts are infected in the first 92 scans of the initially infected host. This occurs with probability 0.4 ($0.4 = (0.99)^{92\cdot}$).

When the initially infected host infects a second host, the second host will wait for 100 milliseconds and then it will send an exploit every 25 milliseconds, just like the first host. The initially infected host will send out four more exploits before the second infected host begins sending out exploits.

After the second host sends out 18 exploits, the threshold for the $LP_{SYN:445}Sum_1$ feature is exceeded. If the second infection occurs on or before the 69th scan of the first host, then the 18th scan of the second host will occur on or before the 91st scan of the first host ($91 = 69 + 4 + 18$) and the $LP_{SYN:445}Sum_2$ feature will detect the presence of the worm before the $LP_{SYN:445}Sum_1$ feature does. Both $LP_{SYN:445}Sum_1$ detection and $LP_{SYN:445}Sum_2$ detection will occur with probability 0.50 ($0.50 = 0.99^{69}$). In other words, $LP_{SYN:445}Sum_1$ detection and $LP_{SYN:445}Sum_2$ detection are equally likely.

We will now sketch the computation of 1) *the expected time to detection*, and 2) *the expected number of infected hosts at the time of infection* both for the case of $LP_{SYN:445}Sum_1$ detection and for the case of $LP_{SYN:445}Sum_2$ detection. These results will then be averaged to obtain 1) *the overall expected time to detection* and 2) *the overall expected number of infected hosts at the time of infection*.

Case 1) $LP_{SYN:445}Sum_1$ **Detection**

In this case, the initially infected host has sent out 92 exploits before the $LP_{SYN:445}Sum_1$ alarm is sounded. This implies an upper bound of 2.3 seconds for the worm detection time ($2.3 = 92 \times 0.025$).

In the case of $LP_{SYN:445}Sum_1$ detection, no secondary hosts are infected in the first 69 scans by the initially infected host. This leaves 23 scans for the initially infected host to infect a second host before the $LP_{SYN:445}Sum_1$ alarm is sounded ($23 = 92 - 69$). The expected number of secondary hosts infected by the initial host in these 23 scans is 0.23. A secondary host has an average of 10 scans to infect a tertiary host before the $LP_{SYN:445}Sum_1$ alarm is sounded ($10 = (23 - 4)/2$). The expected number of tertiary infections is 0.02 hosts ($0.02 = 0.23 \times 0.1$). On the average, the total number of infected hosts on the 92nd scan in Case 1) is 1.3 ($1.3 = 1 + 0.23 + 0.02$).

Case 2) $LP_{SYN:445}Sum_2$ **Detection**

In this case, there are always at least two infected hosts. The second host is infected somewhere between the 1st and the 69th scan of the first infected host, inclusive. Without the assumption that the second host is infected in the first 69 scans, the probability that the second host is infected on the k^{th} scan is $(q^{k-1})p$, where $p = 0.01$ and $q = 0.99$ and $k = 1, 2, 3, \ldots$. Assuming that the second host is infected in the first $N = 69$ scans, Bayes Theorem implies that the expected number of scans (S) for the second infection is given by the following equation[12].

$$S = \frac{\sum_{k=1}^{N} k(q^{k-1})p}{\sum_{k=1}^{N} (q^{k-1})p} = \frac{1 - (N+1)q^N + Nq^{N+1}}{p(1 - q^N)} \quad (1)$$

Substituting $N = 69$, $p = 0.01$, and $q = 0.99$ into the equation above yields the result that $S = 31$ scans is the expected number of scans for the second infection (assuming that the second infection occurs in the first 69 scans). The $LP_{SYN:445}Sum_2$ alarm is sounded when the initially infected host makes 22 more scans after the second host is infected. So, the initially infected host is expected to have made 53 scans when the $LP_{SYN:445}Sum_2$ alarm is sounded ($53 = 31 + 22$). This implies an expected detection time of 1.3 seconds ($1.3 = 53 \times 0.025$) for Case 2.

After the first secondary infection, the initially infected host has 22 scans to infect more hosts before the $LP_{SYN:445}Sum_2$ alarm is sounded. The expected number of these secondary infections is 0.22. The expected number of tertiary infections from these 0.22 hosts is 0.02. The first secondary infection has 18 scans in which to infect a tertiary host before the $LP_{SYN:445}Sum_2$ alarm is sounded. On the average, 0.18 tertiary hosts will be infected during these 18 scans and these tertiary hosts will infect 0.02 quaternary hosts.

On the average, the total number of infected hosts at the time of an $LP_{SYN:445}Sum_2$ detection is 2.4 ($2.4 = 2 + 0.22 + 0.02 + 0.18 + 0.02$). Combining the results of these two cases, the initially infected host is expected to make 73 scans before the infection is detected ($73 = (92 + 53)/2$), the expected time to detection is 1.8 seconds ($1.8 = 73 \times 0.025$), and the expected number of infected hosts is 1.9 ($1.9 = (1.3 + 2.4)/2$).

6.3 Example 2: 8k Vulnerable Hosts

In this example, all of the target network's 8k hosts are vulnerable and in the Zotob worm's 64k IP address space.

[12]The limit as $n \to \infty$ of this equation when $0 < x < 1$ is found in [4], Section 3.5, Example 3, *The Pascal or geometric distribution.*

The effect of Zotob on the target network is greatly reduced by the small number of vulnerable hosts. If every host on the target network were vulnerable and if the same assumptions are made as in Example 1), then the average time to detection would be reduced from 1.8 s. to 0.75 s. and the average number of hosts at the time of infection increases from 1.9 hosts to 7.7 hosts. This change is less than one might expect from such a dramatic increase in the percentage of vulnerable hosts on the network. The minimum and maximum values also change less than one might expect. The most dramatic change is that even though 0.6 seconds is the minimum reasonable time in which the worm is detected, this minimum time now occurs with probability 0.4 instead of with probability 0.04. This is because the probability of infection on any given scan is now 0.125 whereas in Example 1 it was 0.01. The calculation of these results proceeds exactly as in the case of Example 1. The biggest difference is that the chances of $LP_{SYN:445}Sum_1$ detection is now 10^{-4} instead of 0.5 so that $LP_{SYN:445}Sum_2$ is the dominant form of detection in Example 2.

6.4 Example 3: Hit List

In this example, every exploit results in an infection (e.g., the Zotob worm is using a hit list).

The worst possible case is when the worm infects a new host every time it sends out an exploit. There are two possible sub-cases here. In the first one, the worm author is aware of the thresholds used by the algorithm and makes use of them in the design of his worm. In this case, he can, to some extent, hide the presence of his worm in the DC which creates an unusually high threshold of 79 in the sum at depth three feature conditioned on a SYN event for port 445. The result is that as many as 106 hosts can be infected at detection time and detection will take about 350 milliseconds.

In the second case, either 1) the worm authors do not know the values of the thresholds which define the algorithm's OP, or 2) the DC causing the relatively high depth three threshold of "79" threshold has been white listed. This will reduce the number of infected hosts at detection-time from 106 to about 50 but will not change the time to detection too much.

7 Accuracy Analysis

With the OP described in the appendix, we can obtain an algorithm with a FAR below twice per day and which sounds alarms 22 minutes per day (1.5%) on average. We learned that white-listing the few most chatty hosts for UDP:137 (DCs), we could protect the network from worms that target UDP:137. Our algorithm would not detect an infection on a white-listed host from a worm which only used UDP on port 137 to spread. However, our algorithm could still detect such a worm from traffic emanating from secondary infections. We expect that applying a white-list to TCP:139 and TCP:445 will have a similar effect.

The system also has the ability to white-list ports, although this was unneeded in our current deployment. One might want to white-list the port for a P2P application, for example, to lower the FAR associated with P2P broadcasts. This does not hinder the system's ability to detect worms on other ports.

We also experimented with a few, more sophisticated link predicates, such as the necessary network behaviors for Welchia (ICMP followed by TCP:445 traffic) and found no more than one descendant at depth one. When relaxed to port-agnostic link predicates (i.e., ICMP followed by either a TCP SYN packet or a UDP packet to any port), the results were similar—there was at most one descendant. These are perfectly sensitive to even multimodal worms (as long as the worm uses ICMP for reconnaissance) and are far more accurate. Initial evaluations of multi-port link predicates (e.g., a TCP:445 connection followed by another TCP:8888) and achieved similar results. Although multi-port link predicates are very sensitive and accurate, managing the state space off all possible combinations is impractical. In these cases, the expected number of infections at detection time is one.

In the current system, link predicates must be provided a priori. However, it is surprising that a few, very general and sensitive link predicates are so accurate. The few link predicates we have used would catch any unimodal worm or any (unimodal or multimodal) worm that uses ICMP. Finding ways to dynamically determine a worm's IV and create the associated link predicate (which would be perfectly sensitive and practically perfectly accurate) would relax the constraint that the system must know the link predicates a priori and would enable the system to detect more sophisticated worms (including multimodal worms). Exploring this problem space is held for future work.

8 Conclusion & Future Work

The graphical approach we evaluated is more sensitive than alternative approaches discussed in the literature while maintaining an acceptable false alarm rate on the target network. The approach is even sensitive to hit-list, topological, and meta-server TAFs which are the most elusive. Work in progress at the time of this writing indicates that this approach may be extended to quickly detect significant classes of multimodal worms with comparable accuracy to that reported in this paper.

We plan on extending the number of subnets monitored on the target network.

This will enable us to better evaluate descendant trees across an entire enterprise as opposed to large subsections. We will also stand up the detection capability as a real-time system and evaluate the trade-offs of sensitivity and accuracy against performance.

Finally, we have a worm emulation system, which takes a worm specification as input and generates worm-like traffic across an operational network. It will be used to test the sensitivity, and performance of both the offline and the real-time detection capabilities presented in this paper.

References

[1] Dan Ellis, John Aiken, Kira Attwood, and Scott Tenaglia. A behavioral approach to worm detection. In *Proceedings of the ACM Conference on Computer and Communication Security (ACM CCS), Workshop On Rapid Malcode (WORM)*, Washington, DC, October 2004.

[2] F-Secure. http://www.f-secure.com/v-descs/zotob_a.shtml, August 2005.

[3] Brad Karp and Hyang-Ah Kim. Autograph: Toward automated, distributed worm signature detection. In *13th USENIX Security Symposium*, Oakland, CA, August 2004. USENIX.

[4] Harry Lass and Peter Gottlieb. *Probability and Statistics*. Addison-Wesley Publishing Company, Inc., Menlo Park, California, 1971.

[5] David Moore, Vern Paxson, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. *IEEE Security & Privacy*, pages 33–39, July/August 2003.

[6] James Newsome, Brad Karp, and Dawn Song. Polygraph: Automatically generating signatures for polymorphic worms. In *IEEE Symposium on Security and Privacy*, Oakland, California, May 2005. IEEE.

[7] Nicholas Weaver and Stuart Staniford and Vern Paxson. Very fast containment of scanning worms. In *13th USENIX Security Symposium*, Oakland, CA, August 2004. USENIX.

[8] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *6th ACM/USENIX Symposium on Operating System Design and Implementation (OSDI)*, San Francisco, CA, December 2004.

[9] Stuart Staniford. Containment of Scanning Worms in Enterprise Networks. *Journal of Computer Security*, 2004.

[10] Stuart Staniford, David Moore, Vern Paxson, and Nicholas Weaver. The top speed of flash worms. In *Proceedings of the ACM Conference on Computer and Communication Security (ACM CCS), Workshop On Rapid Malcode (WORM)*, Washington, DC, October 2004.

[11] S. Staniford-Chen, R. Crawford, M. Dilger, J. FRANK, J. Hoagland, K. Levitt, and D. Zerkle. GrIDS: A Graph-Based Intrusion Detection System for Large Networks. In *Proceedings of the 19th National Information Systems Security Conference*, pages 361–370, October 1996.

[12] Stuart Staniford and Vern Paxson and Nicholas Weaver. How to 0wn the Internet in Your Spare Time. In *11th USENIX Security Symposium*. USENIX, August 2002.

[13] Symantec. http://securityresponse.symantec.com/avcenter/venc/data/perl.santy.html, December 2004.

[14] Nicholas Weaver, Dan Ellis, Stuart Staniford, and Vern Paxson. Worms verses perimeters: The case for hard LANs. In *12th Annual IEEE Symposium on High Performance Interconnects (Hot Interconnects)*, August 2004.

[15] Matthew M Williamson. Throttling Viruses: Restricting Propagation to Defeat Mobile Malicious Code. In *18th Annual Computer Security Applications Conference (ACSAC)*, San Diego, California, December 2002. Applied Computer Security Associates.

[16] Yinglian Xie, Vyas Sekar, David A. Maltz, Michael K. Reiter, and Hui Zhang. Worm origin identification using random moonwalks. In *IEEE Symposium on Security and Privacy*, Oakland, California, May 2005. IEEE.

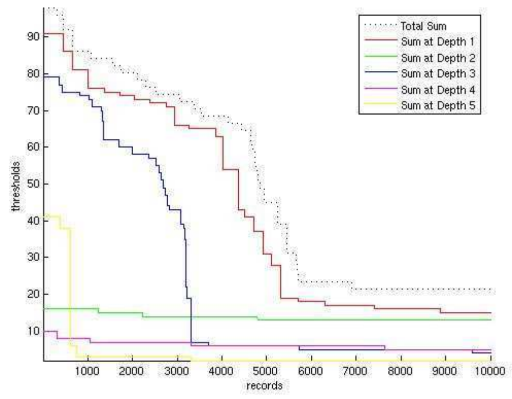

Figure 1: $LP_{SYN:445}Sum_i$: The sum at depth measurements are sorted in descending order

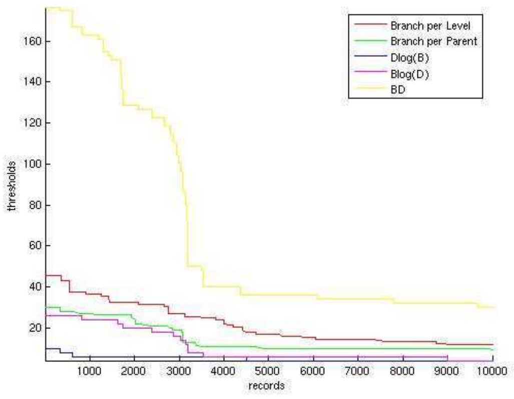

Figure 2: $LP_{SYN:445}$: The branching factor measurements are sorted in descending order

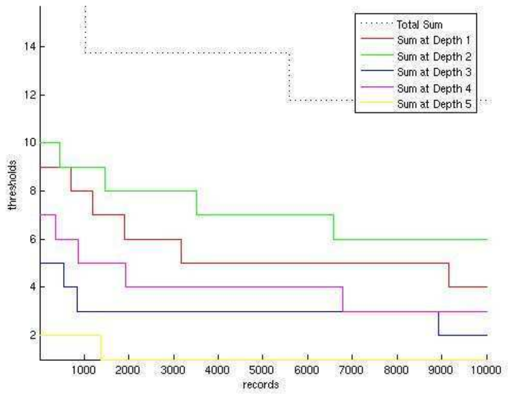

Figure 3: $LP_{UDP:137}Sum_i$: The sum at depth measurements are sorted in descending order

Appendix: Operating Point for 300-Second Feature Window

Key

Link Predicates	Protocol & Flags	Direction	Port Specific
1	UDP	To	Yes
2	TCP SYN	To	Yes

LP[Port]	Sum	Depth	AvgBpL	AvgBpP	SumDlogB	SumBlogD	SumB*D	Sum_i							TTD_i (ms)						
1[135]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	1412	60000	60000	60000	60000	60000	60000
1[137]	18	7	9.02	5.176	6	6	32	9	10	5	7	3	3	2	NaN	NaN	353	1176	7647	60000	60000
1[138]	20	4	14.118	13.059	4	4	24	14	13	11	2	2	2	2	NaN	0	60000	60000	60000	60000	60000
1[139]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	1647	60000	60000	60000	60000	60000	60000
1[1434]	10	2	6.275	0.235	4	4	4	7	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000
1[194]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	1176	60000	60000	60000	60000	60000	60000
1[2049]	6	2	3.137	2	4	4	4	3	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000
1[22]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	706	60000	60000	60000	60000	60000	60000
1[25]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	4706	60000	60000	60000	60000	60000	60000
1[443]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	941	60000	60000	60000	60000	60000	60000
1[445]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	7294	60000	60000	60000	60000	60000	60000
1[6346]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	1412	60000	60000	60000	60000	60000	60000
1[6669]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	7294	60000	60000	60000	60000	60000	60000
1[6699]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	235	60000	60000	60000	60000	60000	60000
1[6881]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	17882	60000	60000	60000	60000	60000	60000
1[7000]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	9176	60000	60000	60000	60000	60000	60000
1[80]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	471	60000	60000	60000	60000	60000	60000
1[9911]	6	2	1.373	0.235	4	4	4	2	2	2	2	2	2	2	24706	60000	60000	60000	60000	60000	60000
2[135]	8	2	5.098	0.235	4	4	4	5	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000
2[137]	4	2	0.392	0.235	4	4	4	2	2	2	2	2	2	2	60000	60000	60000	60000	60000	60000	60000
2[138]	4	2	0.392	0.235	4	4	4	2	2	2	2	2	2	2	60000	60000	60000	60000	60000	60000	60000
2[139]	114	7	NaN	NaN	12	56	373	109	15	92	10	89	4	2	NaN	0	353	8000	24941	60000	60000
2[1434]	6	2	3.137	1.059	4	4	4	3	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000
2[194]	4	2	0.392	0.235	4	4	4	2	2	2	2	2	2	2	60000	60000	60000	60000	60000	60000	60000
2[2049]	6	2	3.137	0.235	4	4	4	3	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000
2[22]	8	2	4.118	1.059	4	4	4	4	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000
2[25]	73	2	50	0.235	4	4	4	70	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000
2[443]	18	2	15.098	0.235	4	4	4	15	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000
2[445]	98	8	45.49	30	10	26	176	91	17	79	10	41	3	2	NaN	0	706	6353	15882	60000	60000
2[6346]	4	2	0.392	0.235	4	4	4	2	2	2	2	2	2	2	60000	60000	60000	60000	60000	60000	60000
2[6669]	4	2	0.392	0.235	4	4	4	2	2	2	2	2	2	2	60000	60000	60000	60000	60000	60000	60000
2[6699]	4	2	0.392	0.235	4	4	4	2	2	2	2	2	2	2	60000	60000	60000	60000	60000	60000	60000
2[6881]	4	2	0.392	0.235	4	4	4	2	2	2	2	2	2	2	60000	60000	60000	60000	60000	60000	60000
2[7000]	4	2	0.392	0.235	4	4	4	2	2	2	2	2	2	2	60000	60000	60000	60000	60000	60000	60000
2[80]	33	3	31.961	4	4	4	8	32	4	3	2	2	2	2	NaN	0	60000	60000	60000	60000	60000
2[9911]	14	2	9.02	0.235	4	4	4	11	2	2	2	2	2	2	NaN	60000	60000	60000	60000	60000	60000

NaN means that no acceptable threshold was found.