Signature Metrics for Accurate and Automated Worm Detection

Prem Gopalan* Mazu Networks
Kyle Jamieson† MIT
Panayiotis Mavrommatis‡ MIT
Massimiliano Poletto§ Mazu Networks

ABSTRACT

This paper presents two simple algorithms, TreeCount and SenderCount, that detect a broad range of exploit-based and email worms, respectively. These algorithms, when combined with automated payload fingerprinting, generate precise worm payload signatures. We show that fundamental traffic properties of most worms, such as infected hosts' attempts to propagate the worm, can serve to detect signatures of non-polymorphic worms reliably and rapidly.

Our prototype monitored over 200 Mb/s of university traffic for 3 months. TreeCount generated new signatures during the Zotob outbreak with no false positives, and also identified known worms like Sasser and Phatbot. SenderCount identified email worms and a spam cluster, while generating ~ 2 false positives/hour.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: Security and protection; C.2.3 [**Network Operations**]: Network monitoring

General Terms

Security

Keywords

network worms, worm signatures, traffic analysis

1. INTRODUCTION

Worms are stealthier and faster than ever. Some, like Slammer, infected most of their vulnerable population in minutes. Moore et al. [13] show that effective containment of a rapid worm can require reaction times as low as a couple of minutes, and that dropping packets that match *worm payload signatures*—byte sequences in packet payloads that identify the worm—is more effective than filtering individual IP addresses. Such signatures, if available, can be fed to firewalls and other network filters to control worm propagation.

The first step in an automated worm defense system is signature generation. Signatures should be *precise*: they should identify only the parts of the worm payload that characterize the worm and do not occur in legitimate traffic. Filtering a benign or non-worm-specific signature may shut down a critical service.

In this paper, we focus on detection and automatic generation of precise payload signatures of exploit-based and email worms. We restrict ourselves to non-polymorphic worms, ones whose payload is largely invariant, because polymorphic worms are not as amenable to payload filtering. While polymorphic worms pose a growing threat, most worms to date have been non-polymorphic. Despite their relative technical simplicity, non-polymorphic worms continue to cause damage, and detecting them accurately and rapidly is a matter of practical importance.

Our work builds on the Earlybird system by Singh et al. [18], as well as behavioral worm detection such as that described by Ellis et al. [8]. We present two algorithms that identify worms and their signatures by measuring traffic properties of those signatures. For example, to detect exploit-based worms, we first identify frequently occurring signatures on the network; then, for each signature, we count the number of unique destinations it was sent to and the number of hops it has made (how far it has propagated). We employ similar *signature metrics* in our email worm detection algorithm.

Some researchers have proposed classifying flows as suspicious and then extracting worm signatures. This method is prone to *false negatives* caused by the classification process. Like Earlybird and behavioral analysis, our algorithms use the fundamental spreading dynamics of a worm and do not depend on *a priori* classification of flows. However, Earlybird's authors report *false positives*, which they address using a signature whitelist. We describe a set of signature metrics that isolate worm traffic accurately and require neither pre-classification nor whitelisting.

This paper makes several contributions. First, it identifies the limitations of existing signature generation schemes. Second, it highlights the differences between exploit-based

*Email:prem@mazunetworks.com
†Email:jamieson@csail.mit.edu
‡Email:mavrommatis@gmail.com
§Email:maxp@mazunetworks.com

WORM'06, November 3, 2006, Alexandria, Virginia, USA.

and email worms, and explains why signature generation should happen differently for these two classes of worms. Third, it identifies *signature metrics* that enable precise signature generation for both types of worms. And finally, it presents and evaluates algorithms that employ these metrics to detect both exploit-based and email worms.

2. RELATED WORK

Recent research has produced both *host-based* and *network-based* approaches to worm signature detection. Host-based systems like TaintCheck [15] and Vigilante [7] benefit from knowledge about the state of the host during an attack. They perform dynamic dataflow analysis to track program activity, including buffer overflows and control transfers. Since they identify worm signatures in terms of program behavior rather than packet payload, they remain effective in the presence of polymorphic worms. Despite this advantage, host-based systems create overhead on their hosts and are more difficult to deploy and administer than network-based solutions.

Network-based signature detection systems attempt to identify characteristics uniquely associated with worm traffic. Traditional firewalls and intrusion detection systems, such as Snort [16], match packet payloads against a predefined set of payload signatures. This approach is simple and frequently effective, but suffers both from false positives (it matches individual packets that look malicious, whether or not the network is vulnerable to that particular exploit) and from false negatives (new worms may not be in the signature database yet).

A more recent and promising approach is behavioral analysis, explored by Ellis et al. [8] and developed independently by several network security vendors. Behavioral signatures describe patterns of network traffic that are common across instances of a worm or class of worms, such as sending data from one machine to the next, scanning, tree-like propagation, and servers becoming clients. Behavioral signatures identify broad characteristics that almost any worm must have in order to propagate, so they are less vulnerable than traditional payload signature systems to false negatives in the presence of new worms. Moreover, since they trigger on worm-like traffic patterns rather than on the presence of a single malicious packet, they are less likely to report false positives when a network is not vulnerable to a particular exploit.

Unfortunately, behavioral signatures are not a complete solution. First, peer-to-peer and other common network applications (for example, Windows workgroups) can exhibit worm-like communication patterns that decrease behavioral signatures' sensitivity. Second, the time necessary for a behavior to become detectable may allow the worm to propagate more broadly. Furthermore, some worms, such as email worms, do not have distinct transport-level behavior, and require some amount of payload analysis. Finally, behavioral signatures are not ideally suited for filtering; Moore et al. [13] explain why filtering worm-specific payload fragments is likely to be a more effective way of stopping worms.

Automated worm payload signature generation systems strive to solve the problem of false negatives in the presence of new worms while retaining the filtering benefits of payload signatures. Systems described in the literature include Honeycomb [12], Autograph [10], Earlybird [18], and Polygraph [14]. The first three all focus on *non-polymorphic* worms, which have at least some invariant payload or a limited number of payload variations. Honeycomb uses a honeypot to gather suspicious traffic and searches least common substrings to generate worm signatures. It assumes that all the traffic it sees is suspicious. It operates on a host-based context, rather than analyzing network-wide traffic. Autograph is designed for larger-scale distributed deployments, but relies on a pre-filtering step to identify flows with suspicious scanning behavior. It does not look for signatures in packets that do not match this filter, so it cannot detect certain types of worms, such as email worms and worms that use hit-lists [20, 3]. Earlybird improves on Autograph by eliminating the pre-filtering step and focusing on a scalable, high-performance implementation. We used Earlybird as the basis and inspiration for our own work. Polygraph's ability to detect polymorphic worm signatures may become critical in the future, but some of its more powerful signature types, such as conjunction signatures and Bayes signatures, are not supported on current firewalls, and its algorithms are slower than Earlybird's or those in this paper. For the rest of this paper we will focus on non-polymorphic worms, which continue to be a threat today.

Our work aims to improve Earlybird-like automated generation of worm payload signatures through the use of better metrics for identifying potential worm traffic. Essentially, we combine ideas from behavioral analysis with the signature-generation approach developed in Earlybird to improve the accuracy of our payload signatures and decrease false positives.

3. MOTIVATION

Worm payload signature detection systems face conflicting challenges. They must not generate false positives on legitimate traffic that frequently exhibits worm-like characteristics, yet they must be sensitive to the start of a worm outbreak. It is crucial, therefore, to pick good metrics by which to determine the likelihood that a signature is worm-related.

Earlybird introduces two good metrics, *content prevalence* and *address dispersion*. A worm's invariant content is likely to be prevalent, or occur frequently, on the network. Moreover, it is likely to be dispersed—associated with many IP addresses. If an invariant payload is prevalent, Earlybird begins to track its address dispersion. If dispersion exceeds a threshold, the payload is flagged as a worm signature.

However, Earlybird's efficacy is limited by two drawbacks of its metrics. First, legitimate traffic frequently exhibits content prevalence and address dispersion. Headers in common protocols used by many clients and servers, such as HTTP, may have these properties. Also, large networks may have hundreds of mail servers and thousands of clients, so mail headers and mass email payloads are prevalent and highly dispersed. (Admittedly, when the mass email is spam, the consequences of false positives are less problematic.) Table 1 lists more examples. Second, prevalent and dispersed signatures in worm traffic may be benign. For example, the Zotob [2] worm uses SMB to connect to vulnerable hosts, so SMB protocol strings are prevalent and dispersed on networks it infects. Yet the strings are common in benign traffic, so filtering them would block Windows file sharing, which uses the SMB protocol.

Earlybird's authors acknowledge the potential false positives caused by content prevalence and address dispersion

Service	Signature	# Srcs	# Dsts	Type
HTTP	1) Gecko/20060111 Firefox/1.5.0.1\015\012Accep	243	676	version numbers
HTTP	late\015\012If-Modified-Since: Thu, 02 Mar 200	170	110	timestamps
HTTP	\015\012Connection: Keep-Alive\015\012Cookie: RMID=8	72	53	identifier patterns
HTTP	Yahoo! Slurp; http://help.yahoo.com/help	212	41	web crawlers
SMTP	pest medications based LICENSED online p	96	34	spam mail
SMTP	ny unauthorized review, use, disclosure	33	35	mail footers
SMTP	SMSSMTP 4.1.11.41) with SMTP id M2006030	35	251	identifier patterns
SMB	\002LANMAN1.0\000\002Windows for Workgroups 3.1a\000	196	76	version numbers

Table 1: Benign 40-byte strings that are both prevalent (> 8/min) and dispersed (associated with > 30 sources and destinations), detected on the Stony Brook University network in under 10 minutes. The existence of many such strings makes them hard to whitelist.

in legitimate traffic, and address them in part with static signature whitelists. They use a protocol parser to whitelist strings from well-known protocol headers (HTTP, SMTP) procedurally. They also show that many frequent signatures in common protocols can be whitelisted manually. However, in general, whitelists remain difficult to maintain and require ad-hoc protocol analysis, time, and expertise. And whitelisted signatures may vary with time. For example, the `If-Modified-Since` HTTP header field, followed by today's date, is prevalent and dispersed on many networks, but it cannot be statically whitelisted, because it will change tomorrow. Moreover, some protocols have arbitrarily variable signatures. For instance, payload strings generated by Inktomi's web crawler may appear dispersed because the crawler consists of many hosts [1], and they vary from one Inktomi customer to another (for example, `Yahoo! Slurp`) in an undocumented manner, so comprehensive whitelisting is difficult. Finally, whitelists provide a window of opportunity for worm authors, who can craft worm payloads to match known protocol headers or other common strings that are likely to be whitelisted.

Finally, although we have not measured this effect, it is possible that Earlybird's metrics and use of whitelists could cause its detection speed to be inversely proportional to network size. As the number of clients and servers for a protocol increases, the dispersion of prevalent and invariant protocol strings increases too: a larger network *exposes* more prevalent signatures that may need to be whitelisted. Since the whitelist may not be updated for all signatures, one could decrease false positives by increasing the dispersion threshold. But increasing that threshold increases the number of hosts that must be probed before the worm is detected, decreasing detection speed.

4. ALGORITHMS

Like Earlybird, we first identify prevalent signatures, using a scheme based on Rabin fingerprints [4]. Formally, a signature is a tuple $s = (d, p)$, where d is a destination port that identifies the exploited service, and p is an invariant payload substring.

We then apply TreeCount and SenderCount to the prevalent signatures to identify worm signatures. We designed the algorithms to meet three goals:

- **Precision.** Worm signatures should differentiate worms from normal traffic, and not include benign substrings of worm payloads.
- **Detection speed.** The algorithms should have speed comparable to that of address dispersion.
- **No signature whitelists.**

The algorithms rely on traffic properties of worm signatures to distinguish those signatures from benign prevalent ones. We quantify these differences by means of *signature metrics*, packet-based measurements associated with signatures. Our algorithms use these metrics to decide whether a signature is worm-related.

We address *exploit-based worms*—which propagate automatically by searching for software vulnerabilities—separately from *email worms* because they differ in 3 important ways.

First, they exhibit different connection behavior. Exploit-based worms make many connection attempts to several hosts, whether to search for new vulnerable hosts or to infect known vulnerable hosts. Email worms, by contrast, do not need to make direct connection attempts to their victims. They rely on users to activate them, and propagate from inbox to inbox via mail servers, without opening direct connections to new victims.

Second, the two types of worms propagate in different address spaces. Whereas exploit-based worms propagate from IP address to IP address, email worms propagate over an overlay network of email clients and servers. This network cannot be understood without application-layer analysis of email headers: at the transport layer, the worm behaves much like normal email traffic.

Finally, because they require human activity, email worms operate on longer time scales than exploit-based worms. For example, prevalent signatures may need to be measured over an interval of hours, rather than minutes, before the payload of an email worm is detected.

4.1 Signature Metrics for Exploit-Based Worms

First, note that once an exploit-based worm infects a host, it either attempts to find new victims or connects to a hitlist of known victims. In either case, the host tries to send the worm payload or exploit to many other hosts (often, in a short interval). This behavior is key to worm propagation. Hence, our first metric simply counts the number of destinations for every packet with signature s sent by *src*, including failed connection attempts, since not all infection attempts may succeed past the target discovery phase.

Metric 1. For a given source address *src* and signature $s = (d, p)$, the *connection fanout*, $fanout(src, s)$, is the number of distinct destination addresses that received either packets matching s or unsuccessful connection attempts from *src*.

Given a prevalent signature s, TreeCount measures the fraction of hosts h that sent s for which $fanout(h, s)$ exceeds

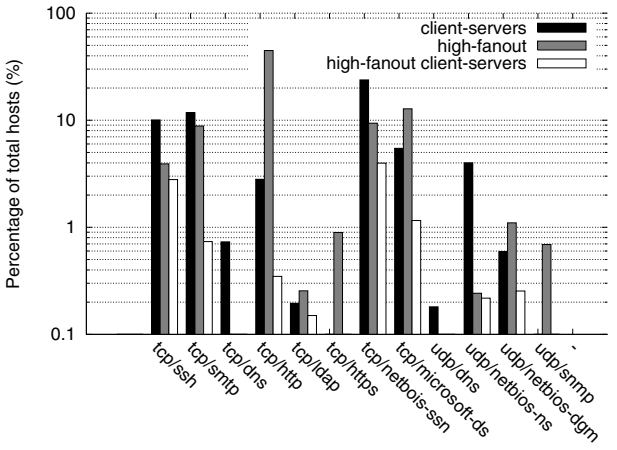


Figure 1: Percentage of clients with high fanout (*high-fanout*), hosts exhibiting both client and server behavior on one port (*client-servers*), and hosts exhibiting both (*high-fanout client-servers*) for any signature for common TCP and UDP services, on a 60,000-host enterprise network. Fanout threshold is 5.

a threshold (*FanoutThresh*). TreeCount never considers s part of a worm unless this fraction exceeds a second threshold (*FanoutRatioThresh*).

Second, observe that a host becomes infected by acting as a server on port d, but then infects other hosts by becoming a client, attempting to connect to port d on those hosts. This alternating client-server behavior is used by Ellis et al. [8] to create behavioral signatures. It is common in exploit-based worms, and we use it to detect the propagation of non-polymorphic worms.

Consider the "first infection" directed graph $G = (V, E)$ for a signature s. V is the set of hosts that send or receive s. A directed edge (i, j) exists if i is the *first* host observed to send s to j, and j has not previously sent s to other hosts. Since only the first infection attempt is recorded, G is acyclic. The *depth* of a node i in a directed acyclic graph (DAG) is the length of the longest path between i and a minimal node [17] (informally, a root of the DAG.) The DAG construction ensures that the DAG's depth is not arbitrarily increased by duplicate infection attempts on nodes with depth 0. This graph is a refinement of the "causal tree" in the work of Xie et al. [22], because it is parameterized on s.

As infected servers become clients and infect other servers, the number of nodes with depth greater than 1 increases and the maximum depth across all nodes also increases. A prevalent signature's maximum depth, therefore, measures the propagation of that signature in a network. This metric is relatively low (1 or 2) for traffic where most hosts are either clients or servers, such as HTTP, and higher than normal (≥ 2) for worms:

Metric 2. For a given signature $s = (d, p)$, the *maximum propagation depth*, $mpd(s)$, is the maximum depth of a node in the DAG constructed by the first infection attempts with packets matching s.

TreeCount never considers a signature s part of a worm unless $mpd(s)$ exceeds a threshold (*DepthThresh*). To avoid rare false positives (for example, when hosts connect via a proxy), it further qualifies the MPD metric with a *minimum breadth* threshold inspired by worms' connection fanout: there must be at least *BreadthThresh* hosts at each level of depth of the first-infection DAG.

Fanout and propagation depth must distinguish worm traffic from benign, non-worm traffic. In particular, we expect the fraction of hosts that have high fanout for any benign prevalent signature $s = (d, p)$ to be low, and the fraction of hosts that exhibit both client and server behavior on port d to be low also. Consequently, the fraction of hosts that exhibit both properties for a benign signature should be even smaller. Figure 1 shows that this is indeed the case, based on flow records collected over a few weeks in early 2006 on a busy enterprise network with over 60,000 hosts (including 55,000 HTTP clients and 37,000 SMB clients).

Of course, many worms are multimode: they attempt to exploit multiple vulnerabilities, possibly on different ports. However, they typically probe for vulnerabilities on all of these ports, until an exploit succeeds, so we expect to detect signatures with high fanout and high propagation depth on each of these ports. Our algorithm will usually generate multiple signatures in this case. We then consolidate the signatures into fewer alerts based on the fraction of infected hosts in common. This occurred, for example, during detection of an outbreak of Spybot, which propagates over ports TCP/135 and TCP/445, among others.

Algorithm. The TreeCount algorithm operates on packets that contain prevalent signatures. It takes as input the prevalent signature, s, and the packet's source and destination addresses. It measures the fraction of hosts that have high fanout (fanout exceeding a given threshold) for s, and updates a representation of the DAG that is used to measure s's maximum propagation length. If both metrics exceed given thresholds, the algorithm classifies the signature as worm-related and generates an alert. See Algorithm 4.1 for details.

Although the pseudocode contains some linear-time operations for clarity, in real life TreeCount processes each prevalent signature in a packet in constant time. The host sets, including source and destination hosts, are implemented as hash tables, and the fanout ratio and maximum propagation depth are computed incrementally in constant time.

4.2 Signature Metrics for Email Worms

Many destructive email worms, such as Sobig, MyDoom, and Netsky, were non-polymorphic. When such a worm infects mail clients on a network, clients begin to send email messages with similar bodies or attachments.

Our first metric for email worms, therefore, simply counts the clients on the monitored network that send email (TCP/25) packets with the same signature. We count only internal clients on the monitored network to avoid detecting incoming spam. In addition, note that a single mail message may be relayed by several *intermediate mail servers* before reaching its destination, so that one message may appear to have many sources, though only one of these is the real mail client and the others are relays. To count the real mail clients, we resort to whitelisting the monitored network's SMTP servers and never count them as mail clients. (For reasons that will become clear in Section 5, we also whitelist legitimate automated mailers, such as library notification systems.) Fortunately, even on large networks the set of mail servers and automated mailers is known and manageable, so listing mail servers in this way does not violate our requirement for practicality.

Algorithm 4.1: TREECOUNT(s, src, dst)

procedure FANOUT(h)
return $(|dsts(h, s)|)$

procedure FANOUT-RATIO()
$f \leftarrow \{h$ S.T. $h \in srcs(s) \wedge$ FANOUT$(h) > FanoutThresh\}$
return $(|f|/|srcs(s)|)$

procedure MPD()
return $(max(\{depth(h)$ S.T. $h \in hosts(s)\}))$

procedure MIN-BREADTH()
return $(min(\{|level(d)|$ S.T. $0 < d \leq$ MPD$()\}))$

procedure UPDATE-DEPTH()
if $src \in hosts(s)$
  then { **if** $dst \notin hosts(s)$ **then** $depth(dst) \leftarrow depth(src) + 1$
  else { $depth(src) \leftarrow 0$; **if** $dst \notin hosts(s)$ **then** $depth(dst) \leftarrow 1$
$hosts(s) \leftarrow hosts(s) \cup \{src, dst\}$
$level(depth(dst)) \leftarrow level(depth(dst)) \cup \{dst\}$

main
UPDATE-DEPTH()
$srcs(s) \leftarrow srcs(s) \cup \{src\}$
$dsts(src, s) \leftarrow dsts(src, s) \cup \{dst\}$
if (MPD() $> DepthThresh$ **and**
  FANOUT-RATIO() $> FanoutRatioThresh$ **and**
  MIN-BREADTH() $> BreadthThresh$)
then output (WORMALERT(s)))

Metric 1. Given a signature $s = (TCP/25, p)$, its *client dispersion*, $cdisp(s)$, is the number of mail clients inside the monitored network that send packets matching s and are not well-known mail servers.

The second metric is based on the observation that most email worms send many messages in a short time. In particular, possibly due to configurable limits on the number of recipients, most email worms do not send a single message to many recipients, but rather many messages at once. By contrast, only a small fraction of normal mail messages are sent at a high rate (> 1/min), and mail traffic is bursty, with long periods of inactivity [21].

Algorithm 4.2: SENDERCOUNT(s, src, dst)

procedure CDISP()
return $(|clients(s)|)$

procedure AVG-NSENT()
return $(average(\{nsent(s, h)$ S.T. $h \in clients(s)\}))$

procedure AVG-NSERVERS()
return $(average(\{|servers(s, h)|$ S.T. $h \in clients(s)\}))$

main
$clients(s) \leftarrow clients(s) \cup \{src\} - smtpservers$
$servers(s, src) \leftarrow servers(s, src) \cup \{dst\}$
$nsent(s, src) \leftarrow nsent(s, src) + 1$
if CDISP() $> ClientsThresh$ **and**
  (AVG-NSENT() $> AvgNSentThresh$ **or**
  AVG-NSERVERS() $> AvgNServersThresh$)
then output (WORMALERT())

Metric 2. Given a signature $s = (TCP/25, p)$ and a mail client *src*, the *sent packet count*, $nsent(src, s)$, is the number of packets sent by *src* that match s.

The third metric for detecting email worms stems from the observation that most worms carry their own SMTP engines and usually contact many SMTP servers, including external ones. Carrying its own SMTP client allows a worm to spread without relying on the infected machine's software. But the worm's SMTP client must then learn about SMTP servers on the network, and while it may be possible to examine the local SMTP configuration, it is usually easier and more effective to use brute force and attempt to contact numerous (> 5) SMTP servers. By contrast, a normal mail client typically contacts no more than 2 or 3 SMTP servers.

Metric 3. Given a signature $s = (TCP/25, p)$ and a mail client *src*, the *SMTP server count*, $nservers(src, s)$, is the number of SMTP servers to which *src* sends packets that match s.

Algorithm. The SENDERCOUNT algorithm operates on SMTP packets that contain prevalent signatures. Its inputs are the signature and the source and destination addresses of the packet. It tracks these three metrics and alerts when a signature exceeds a combination of them. See Algorithm 4.2 for details.

Like TREECOUNT, SENDERCOUNT operates in constant time on each signature. The *clients*, *servers*, and *nsent* sets are implemented as hash tables, and the AVG-NSENT and AVG-NSEVERS counts can be computed incrementally, using constant time for each signature.

5. EVALUATION

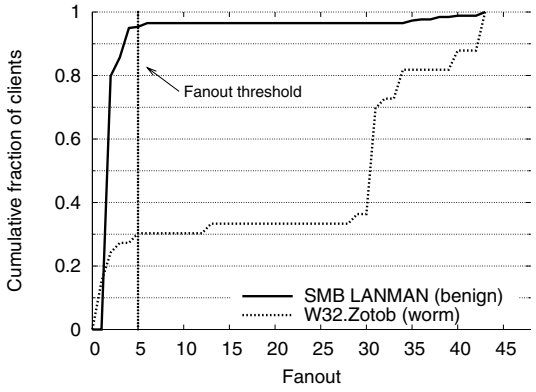
We implemented a prototype to evaluate our algorithms' performance. The system ran at Stony Brook University, for 3 months, seeing an average of 200 Mb/s. It obtained data from mirror ports on two core campus switches, one connected to the campus network and the other to the residential network (ResNet). It captured traffic between campus facilities, as well as ResNet traffic transiting into and out of the campus network.

5.1 System Design and Tuning

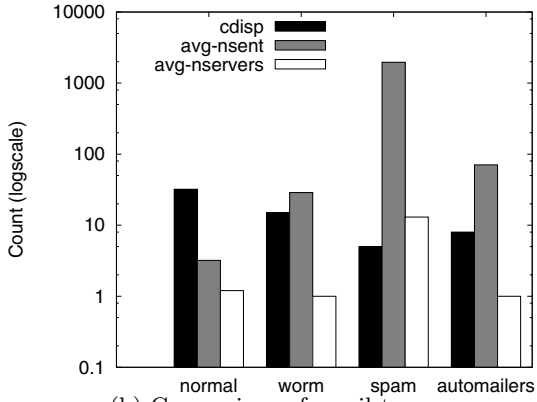
Our prototype worm sensor sifts through packets to find prevalent signatures, applies one of our two algorithms to packets that contain prevalent signatures (SENDERCOUNT for TCP/25 packets, TREECOUNT for all others), and consolidates worm signatures into alerts. It also compares signatures against a list of Snort [16] rules to name known worms.

The code is written in C++ as part of a single-threaded user-level Click [11] configuration. We use a dual 2.6 GHz Intel Xeon server with two Intel PRO/1000 ethernet cards, running Linux 2.4.

We employ Broder's version of the Rabin fingerprinting algorithm [4] and value sampling like Earlybird to compute signature prevalence [18]. We generate fingerprints of 40-byte substrings for most packets, but fingerprints of 200-byte substrings for SMTP traffic. (We refer to the length of these substrings as the "fingerprint length".) Given that fingerprints are computed for all substrings of length l, each packet with a payload of n bytes produces $n - l + 1$ fingerprints. The longer substrings that we use for SMTP traffic are more expensive to compute. However, SMTP



(a) Fanout distribution for Zotob and benign SMB signatures in worm traffic.



(b) Comparison of email types.

Figure 2: Sample parameter tuning data.

is only a small fraction of traffic, and the longer substrings reduce false positives caused by long invariant content, such as SMTP headers, that appears both in email worms and normal email.

We maintain two prevalence tables [18]: one for non-SMTP fingerprints, garbage-collected after 3 minutes of inactivity, and one for SMTP fingerprints, where the timeout is 30 minutes. Each table stores up to 10 million fingerprints. To implement prevalence tables, we evaluated both traditional Bloom filters [5] and the Click toolkit's hashmap [11]. Our Bloom filter takes a 64-bit fingerprint as input and uses 2 independent hash functions. The filter has a false positive rate of 0.5% and requires 256MB for 10 million fingerprints. By comparison, a hashmap implementation requires 320MB and is 5-10% slower. Nonetheless, we chose the hashmap because it produces no false positives.

The algorithms measure each metric over discrete time periods: when a period ends, all metrics and associated data are reset. This simple method works well in practice. We ran experiments to pick practical values for the measurement period: 20-30 minutes for TreeCount, and 2-3 hours for SenderCount. The SenderCount period is longer because email worms propagate more slowly than scanning worms. The algorithms are not very sensitive to the exact duration of these measurement windows. The choice of measurement window is a function of the worm's propagation speed, the amount of packet sampling (if any) performed by the system, and the memory available. Of course, if a worm does not exhibit propagation behavior within the specified measurement window, our algorithms will not detect it.

After picking reasonable thresholds, we refined them by studying different types of traffic. For example, Figure 2(a) shows the distribution of connection fanout for a Zotob-specific signature and a benign SMB signature that also appears in Zotob traffic. Since there is a large amount of legitimate background SMB traffic, the benign SMB signature has fanout ≤ 5 on almost all ($> 90\%$) hosts, whereas the Zotob-specific signature has fanout ≥ 30 on over 60% of hosts. Analyses such as this led us to pick *FanoutThresh* $= 5$ and *FanoutRatioThresh* $= 0.8$. Likewise, after examining, among other things, hosts' client-server behavior (see Figure 1) and TreeCount's false positive rate (discussed further in Section 5.2), we picked *DepthThresh* $= 2$ and *BreadthThresh* $= 3$. With these settings, it is theoretically possible for TreeCount to detect a worm outbreak after as few as 2 successful infections (A infects B, B infects C, and both A and B scan other hosts that are not vulnerable). In reality, the number of both scans and infections is likely to be substantially higher in order for a signature of the exploit to become prevalent.

Similarly, we studied email traffic to refine thresholds for SenderCount. Figure 2(b) shows the 3 email metrics for 4 example signatures of different types: normal email, an email worm (W32.Swen.A), outgoing spam, and automated mailers. Benign signatures have high client dispersion in normal email, but their average sent packet count (*avg-nsent*) and SMTP server count (*avg-nservers*) are low. Email worms frequently have high *avg-nservers*, but Swen in particular does not, so it is not distinguishable from automated mailers in this example; hence the need to whitelist automated mailers, as mentioned in Section 4.2. Outbound spam traffic is also not distinguishable from many worms, but this is not a huge drawback: network administrators are happy to detect spam clusters. (Note that inbound spam generates no alerts because *cdisp* counts only internal clients.) Ultimately, we picked *ClientsThresh* $= 5$, *AvgNSentThresh* $= 10$/hour, and *AvgNServersThresh* $= 5$/hour.

We are in the process of deploying this system on additional networks. While it is still early to say definitively, we believe that the constants and thresholds that we have picked reflect general properties of worm traffic, rather than of specific networks. As a result, we do not expect to repeat the same tuning process on other networks.

5.2 Experience and Measurements

In daily use, TreeCount and SenderCount have proved reliable and effective. TreeCount ran for days during the summer and fall of 2005 with no false positives. It caught the first Zotob outbreak at Stony Brook University on August 16-18, 2005, and much other worm activity, including Sasser, Phatbot, and Spybot. SenderCount generated some false alerts for mass email (department email, library notifications), leading us to whitelist legitimate mass mailers as well as local mail servers. But it also detected email worms, such as Swen, as well as several hosts controlled by a backdoor and used to send spam.

We also used traces of Stony Brook University traffic to evaluate different parameter values and compare our metrics to the address dispersion metric used by Earlybird.

False positives. Figure 3(a) plots false positives over time for several address dispersion and TreeCount thresholds. (t in the legend denotes the address dispersion threshold; f: fanout threshold; d: depth threshold; b: breadth threshold.) The input traffic contains no worms to the best of our knowledge, and the generated signatures match no known Snort rule, so we believe they are all false positives. In 15 minutes, address dispersion generates $\sim$ 180 signatures using $t = 10$, and 35 signatures with $t = 30$. Using propagation depth constrained by a breadth threshold of 2 ($f = 0, b = 2$), TreeCount produces 64 signatures. As we increase b and then f, ultimately to our chosen values of 3 and 0.8 respectively, the false positives decrease and then disappear. Similarly, Figure 3(b) shows data for email traffic, varying fingerprint length. With our chosen thresholds for each metric, SenderCount outperforms address dispersion. Moreover, with a longer fingerprint (200 bytes), it generates no false positives. Of course, longer fingerprints assume that malicious messages or attachments have considerable invariant content, but (after checking against address dispersion and shorter fingerprints) we are not aware of false negatives.

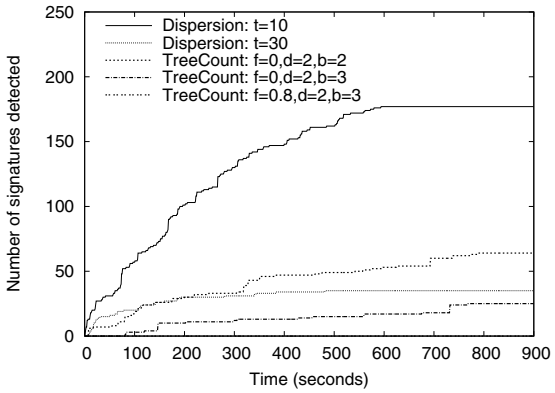
Sensitivity. Figure 3(c) compares the performance of address dispersion and TreeCount on SMB traffic that includes the Sasser/Korgo worm. Each point represents the generation of one or more signatures. The y-axis denotes sensitivity, a measure of the likelihood that the signature is a true positive. Sasser scans at least 64 hosts at a time on port TCP/445, so we (conservatively) define sensitivity as the percentage of hosts associated with a signature that scan more than 16 hosts on that port. TreeCount detects the first true positive more slowly than address dispersion, but address dispersion (especially with $t = 30$, the threshold recommended in the Earlybird paper) also detects several false positives (low sensitivity), whereas TreeCount does not.

Detection speed. Relative to Earlybird, TreeCount and SenderCount may sacrifice detection speed in favor of fewer false positives: their metrics look for more structure in the traffic than just address dispersion. Preliminary results from an analytical worm model [6] indicate that the detection time for our algorithms is roughly constant, independently of the size of a worm's hit-list. Earlybird, by contrast, takes longer on worms with small hit-lists, and less time on worms with large ones (a signature becomes highly dispersed immediately). In practice, both schemes appear to generate signatures very quickly, often within seconds of the first infection.

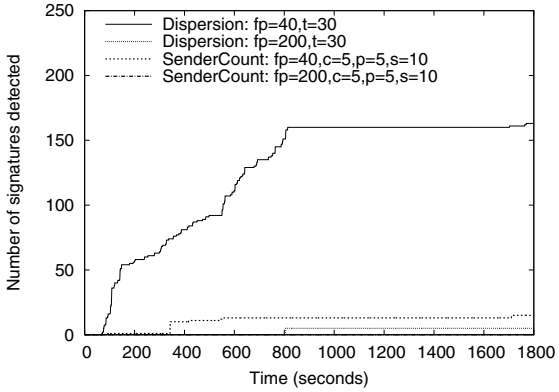
6. DISCUSSION

Sensitivity analysis. Our empirical experience shows that TreeCount and SenderCount are quite accurate: they generate very few false positives and (as best we could determine) false negatives. Nonetheless, several potential issues bear further investigation.

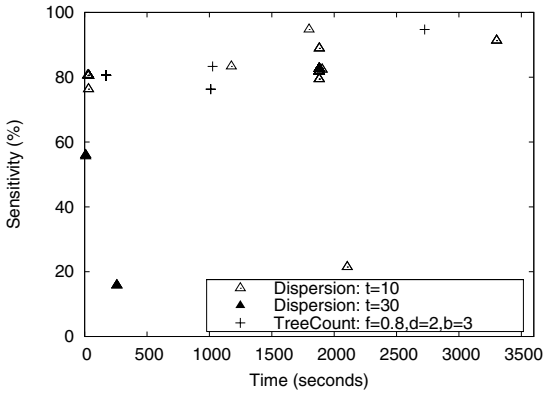
First, peer-to-peer applications such as BitTorrent exhibit behavior that might satisfy our metrics: hosts have many peers, they act as both clients and servers, and payload propagates in a chain from one host to the next. While our system saw much BitTorrent activity at Stony Brook and (correctly) never identified it as a worm, we must further



(a) False positives: signatures detected in normal traffic.



(b) Email false positives: signatures detected in normal email.



(c) Sensitivity: comparison to address dispersion for Sasser/Korgo signatures. Each point may represent multiple detected signatures.

Figure 3: False positives and true positives for TreeCount/SenderCount and address dispersion. **t in the legend denotes the address dispersion threshold; f: fanout threshold; d: depth threshold; b: breadth threshold; fp: fingerprint length; c: client dispersion threshold; p: *avg-nservers* threshold; s: *avg-nsent* threshold.**

characterize the differences between worms and P2P traffic to ensure that false positives do not occur.

An opposite concern is false negatives caused by an incomplete view of the worm propagation tree, due to sampling or patchy monitoring. We consider this problem to be unlikely in practice. Any losses due to sampling are likely overcome by the repetitive nature of worm traffic. And while a poor choice of monitoring location may delay worm detection, our threshold levels allow the algorithms to alert after seeing only a small subset of the worm propagation tree.

Finally, we are vulnerable to some of the same evasion concerns raised by the Earlybird authors. A worm writer may use well-known legitimate strings to create collateral damage when a signature is used for worm containment. He may use address spoofing to hide worm propagation when only one packet is required for infection. And he may use IP fragmentation and other IDS evasion methods to decrease the prevalence of a signature.

Implementation alternatives. We focused on simplicity and acceptable performance in implementing our prototype, but more sophisticated algorithms and data structures may improve the performance of a mature system. For example, threshold random walk [9] and fast algorithms for superspreader detection [19] may help fanout detection scale to higher traffic rates without a corresponding increase in memory requirements.

7. CONCLUSION

TreeCount and SenderCount are two new, simple algorithms that detect and generate worm payload signatures. They use behavioral characteristics of worms' prevalent content, such as connection fanout and propagation depth, to decrease false positives without requiring whitelists of known signatures. Prolonged experience on a campus network shows that the algorithms have good performance and produce fewer false positives than previous schemes. While we have not addressed the challenging problem of polymorphic worms, these algorithms are a pragmatic and effective solution to today's non-polymorphic worms.

Much work remains to be done to optimize our system's performance, to test the algorithms on more networks and traffic types, and to evaluate their effectiveness more precisely. Over time, we may devise new, better metrics. We also hope to apply a similar metrics-based approach to other types of malware, such as botnets.

8. ACKNOWLEDGMENTS

We are indebted to the IT staff of Stony Brook University for access to their network. Many thanks to Glenn Brewer, Jaeyeon Jung, Frans Kaashoek, Xiangjing Chen and Narayanan Sadagopan for discussions and feedback on the paper, as well as to the anonymous referees for their comments and suggestions.

9. REFERENCES

[1] Inktomi web search faq. Technical report. `http://support.inktomi.com/Search_Engine/Product_Info/FAQ/searchfaq.html#slurp`.

[2] Malicious software encyclopedia: Worm:win32/zotob.a. Technical report. `http://www.microsoft.com/security/incident/zotob.mspx`.

[3] The spread of the witty worm, caida. Technical report. `http://www.caida.org/analysis/security/witty`.

[4] A. Broder. Some applications of Rabin's fingerprinting method. *In R. Capocelli et al. eds., Sequences II: Methods in Communications, Security, and Computer Science*, pages 143–152, 1993.

[5] A. Broder and M. Mitzenmacher. Network applications of bloom filters: A survey. In *Proceedings of the Allerton Conference*, 2002.

[6] Z. Chen, L. Gao, and K. Kwiat. Modeling the spread of active worms. In *Proceedings of IEEE INFOCOM*, 2003.

[7] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham. Vigilante: end-to-end containment of internet worms. In *Proceedings of ACM SOSP*, October 2005.

[8] D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia. A behavioral approach to worm detection. In *Proceedings of ACM WORM*, Washington, DC, October 2004.

[9] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proceedings of IEEE Security and Privacy*, Oakland, CA, May 2004.

[10] H. A. Kim and B. Karp. Toward automated, distributed worm signature detection. In *Proceedings of Usenix Security*, San Diego, CA, August 2004.

[11] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kasshoek. The Click modular router. *ACM TOCS*, 18(3):263–297, August 2000.

[12] C. Kreibich and J. Crowcroft. Honeycomb–creating intrusion detection signatures using honeypots. In *Proceedings of ACM HotNets-II*, November 2003.

[13] D. Moore, C. Shannon, G. Voelker, and S. Savage. Requirements for containing self-propagating code. In *Proceedings of IEEE INFOCOM*, April 2003.

[14] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of IEEE Security and Privacy*, May 2005.

[15] J. Newsome and D. Song. Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software. In *Proceedings of NDSS 12*, 2005.

[16] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of Usenix LISA*, 1999.

[17] B. Schroder. *Ordered Sets - An Introduction*. Birkhauser, Boston, 2003.

[18] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of ACM SOSP*, December 2004.

[19] S. Venkataraman, D. Song, P. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *Proceedings of NDSS 12*, 2005.

[20] N. Weaver, V. Paxson, S. Staniford, and R. Cunningham. A taxonomy of computer worms. In *Proceedings of ACM WORM*, 2003.

[21] M. Williamson. Design, implementation and test of an email virus throttle. In *Proceedings of ACSAC 19*, 2003.

[22] Y. Xie, V. Sekar, D. Maltz, M. Reiter, and H. Zhang. Worm origin identification using random moonwalks. In *Proceedings of IEEE Security and Privacy*, 2005.