Polymorphic Worm Detection and Defense: System Design, Experimental Methodology, and Data Resources

Jisheng Wang, Ihab Hamadeh, George Kesidis, and David J. Miller

Departments of EE and CS&E, Penn State University, University Park, PA 16802

jzw128@psu.edu, hamadeh@cse.psu.edu, kesidis@enr.psu.edu and millerdj@ee.psu.edu

ABSTRACT

The polymorphic variety of Internet worms presents a formidable challenge to network intrusion detection and methods designed to extract payload signatures for worm containment. Recently, several systems, including Earlybird and Polygraph, have been proposed, based on efficient processing of payloads to extract signatures that are either explicitly indicative of an attack (exploit code strings) or which have unusual statistical character (content prevalence, address dispersion) consistent with worm activity. While these works are seminal, these systems have limitations that affect accuracy of the extracted signatures and/or practicability of the system's deployment. Earlybird's signature extraction is fragile to polymorphism, while Polygraph makes assumptions about data availability and the accuracy of front-end flow classification. This method also possesses high complexity.

We propose a new method which, fundamentally, integrates header-based multidimensional flow clustering as front-end processing, with content sifting (signature extraction) performed, separately, solely on each cluster in the (small) subset of identified suspicious clusters. Front-end clustering improves purity of the (separate) signature pools and also reduces complexity. We apply a "suffix tree" approach to signature extraction, gleanig both length and frequency information. We demonstrate efficacy of our approach on a (background) trace taken from a /24 in Taiwan, which we salt with worm traffic based on two realistic polymorphic mechanisms that we propose. Since there is a dearth of public data for such testing, we have also made an anonymized version of this trace available, based on randomized headers and *fingerprinted* payloads.

Categories and Subject Descriptors

C.2.3 [Computer-Communication Networks]: Network Operations – *network management*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGCOMM'06 Workshops, September 11-15, 2006, Pisa, Italy.

Copyright 2006 ACM 1-59593-417-0/06/0009...\$5.00.

General Terms

Design, security

Keywords

Polymorphic worms, worm signature extraction, network anomaly detection, intrusion detection system

1. INTRODUCTION

The speed of recent self-propagating (scanning) worms represents a major challenge to the design of effective detection and containment systems. Slammer compromised more than 75,000 hosts within 10 minutes, while Nimda infected more than 2 million machines. [10] has suggested that the response window, in order to effectively curb a fast-scanning attack, is on the order of one minute. The key to worm containment is the blocking of all traffic that matches the worm's (payload) signature. Unfortunately, initial *determination* of this signature has in the past required time-consuming manual inspection of (already) detected worm traffic by network experts – this process can take hours, rather than (the required) seconds. This gap has motivated several recent, ambitious efforts to *automatically* extract worm signatures from Internet traffic, so as to achieve a prompt containment and defense response [14][6][11]. These methods are based on efficient processing of payloads to extract signatures that are either explicitly indicative of an attack (exploit code strings) or which have unusual statistical character (content prevalence, address dispersion) consistent with worm activity. While these works are seminal, there are limitations to each of them relating to the accuracy of the extracted worm signatures, to (high) system complexity, and/or to the assumptions made in designing the system.

In this paper, we first summarize each of these methods, along the way identifying its drawbacks. We then propose a new system for polymorphic worm detection and signature extraction. A key aspect of our approach is to apply, as a *front-end* processing stage, computationally efficient (IP header-based) multidimensional flow clustering [16], customized here to specifically identify suspicious traffic flows. This flow clustering algorithm is highly successful at narrowly localizing individual attacks/traffic anomalies within a small set of traffic clusters with, moreover, distinct attacks/anomalies largely *segregated* from each other (and from innocuous traffic clusters), either in separate clusters or in separate groups of clusters. The purity of these clusters helps to achieve high accuracy of signature extraction, which we perform

separately on each cluster. Moreover, with content sifting (signature extraction) from packet payloads only performed on the (small) subset of traffic contained in the suspicious clusters that we identify, this approach achieves good computational efficiency. While signature extraction is accomplished via Rabin fingerprinting in [14][6] and via the color set size method in [11], we extract signatures using generalized suffix trees [9], gleanings both string length and frequency of occurrence information. These features together are distinctive in identifying accurate worm signatures, and accuracy is further improved if there is an innocuous traffic pool available for assessing false alarm rates for candidate signatures.

While the proposal of an improved defense system is one contribution, we also address the issue of devising a proper experimental methodology for polymorphic worm IDS testing – since polymorphic attacks are at present quite rare in the wild, it was necessary to generate synthetic worm payloads that model polymorphism in a realistic way, and then to appropriately salt the real (background traffic) trace that we used, which was taken from a /24 in Taiwan and primarily represents peer-to-peer traffic. Finally, since there is a dearth of publicly available traces that even contain payloads, let alone worm traffic, and, to our knowledge, no resources for benchmarking polymorphic worm IDSes, we have made an anonymized version of our salted trace available for possible future studies, based on randomization of IP headers and *fingerprinted* (via suffix trees) payloads.

2. PRIOR WORK ON WORM SIGNATURE EXTRACTION

Earlybird [14] in principle performs signature extraction on every observed packet. Their method applies Rabin fingerprinting [12] of fixed length segments, using a sliding window run over the packet payload. In principle, their method generates candidate signatures for all possible substrings of a given length. Content frequency (“prevalence”) is used to reduce the candidate signature pool. Subsequently, “address dispersion” is measured for each signature in the reduced pool. Signatures with both high content prevalence and address dispersion are selected and used to block traffic. While Earlybird in principle performs inspection of every payload, in practice both traffic subsampling and subsampling of signatures is needed to achieve computational feasibility – only 1/64th of all possible substrings are extracted in [14], which can lead to missed detections. Another main drawback is that [14] uses a fixed, long window (40 bytes, possibly corresponding to memory pagination), which can be defeated by simple worm variation – e.g. 1-byte insertions and deletions, or by adding a NOOP every 40 bytes – let alone by significant polymorphism.

Unlike Earlybird, Autograph [6] proposed to apply a front-end flow classifier to first identify packets with suspicious scanning activity (based on failed SYN ACKs, ICMP host unreachable messages) and then to only extract signatures from these packets. They also used Rabin fingerprints, but based on variable-length non-overlapping segments, defined via a content partitioning algorithm. [6] also explored some initial ideas for distributed signature generation, proposing that individual monitors share their suspicious IP address lists. One disadvantage of [6] is that their packet classifier would not work for UDP worms (like Slammer, Witty). Moreover, [6] did not actually implement a flow classifier; thus, how the accuracy of flow classification affects the accuracy (and responsiveness) of signature generation

Figure 1. Structure of payload-based intrusion detection system.

was not explored. Also, while use of variable-length segments is less sensitive to worm variation than a fixed (40-byte) window, [6] partitioned into segments via one or more predefined breakpoints – such partitioning is not likely to work effectively for a diverse set of worms, in particular day-zero worms. Some recent work [13] aims to improve on the partitioning in [6].

[11] focused on extracting signatures for polymorphic worms. Again, front-end flow classification was proposed, to create a suspicious flow pool and an “innocuous” pool. Signatures are extracted from the suspicious pool and then evaluated on the “innocuous” pool to assess the false alarm rate. An efficient algorithm (the color set size method (CSS) [5]) was used to find the substrings that occur at least k times within an n symbol string (in time linear in n). These strings are treated as “tokens”, with token conjunctions taken as the worm signatures. No method was actually developed in [11] for classifying packets as innocuous or suspicious; however the accuracy of signature extraction relies heavily on the purity of both pools. Moreover, while [11] did some experiments to evaluate performance when there are innocuous flows in the suspicious pool, they did not consider suspicious flows within the innocuous pool. Another problem is that even if the suspicious pool is pure worm, it may contain multiple, different worms. Common substrings across different worms – “hybrid worm signatures” – are not likely to be good signatures. A hierarchical clustering algorithm, based on a false alarm criterion, was proposed in [11] which attempts to segregate packets from different worms in different clusters. However, even though this algorithm is greedy, its complexity is considerable and appears to be too great for on-line implementation. Moreover, this method relies on the purity of the innocuous flow pool.

3. NEW POLYMORPHIC WORM IDS

3.1 Introduction

All three methods discussed in the last section have limitations pertaining to (header-based) flow mining – [14] measures destination IP address dispersion, but only *after* having performed deep inspection on every packet (or a subsample). [6] and [11] suggested front-end flow classification to create suspicious and

innocuous pools. However, they neither developed nor evaluated actual flow classifiers. Moreover, flow classification is not the same as flow clustering – one would like not only to separate out a suspicious flow pool, but also separate different attacks/anomalies into distinct clusters, as discussed previously. Multidimensional flow mining based on the IP header [3][16], to identify flows that possess both significant traffic volume and unusual header statistics, provides a natural way to resolve some problems in [14][6][11]. As indicated in Figure 1, we apply flow mining and header-based detection of suspicious flows as the front-end processing. Note that this changes the order compared to [14], with address dispersion first evaluated and deep packet inspection only performed on the suspicious flow subset. This reduces complexity relative to [14] and in principle allows the devotion of greater sophistication to the worm signature extraction (to be applied only to suspicious flows). Our flow mining and detection not only achieve flow classification but also seek to identify separate clusters for each anomaly/attack. Thus, the hierarchical clustering in [11] can be avoided. We believe the CSS method used in [11] is a good alternative to Rabin fingerprinting for extracting signatures. However, our flow mining allows us to directly extract signatures separately for each suspicious cluster, rather than from the entire suspicious flow pool.

In the next subsection, we briefly review the multidimensional flow mining in [3][16], which extracts all (both nominal and suspicious) high-volume, multidimensional clusters. We then specialize [16] to directly mine *only* those clusters that are good candidates for being suspicious – this substantially reduces the time and space complexity of flow mining. Each mined cluster is subsequently evaluated, based on its header statistics, to definitively determine whether it will be treated as innocuous or suspicious, with signature extraction, to be discussed, applied to the latter clusters.

3.2 Mining Suspicious (Worm) Clusters

Identification of frequent item sets in databases is a general-purpose capability that has recently found networking applications [8][3], specifying flows based on the 5-tuple of attributes in the IP header. While this *flow mining* has been primarily used for off-line traffic digesting [3], a key premise in [16][19] is that flow mining can be a core component of anomaly detection on a router – mining, can automatically identify significant, suspicious traffic flows. One can then block traffic that fits this flow’s definition or run more sophisticated detection processing, perhaps change detection [7][19] or, as proposed here, worm signature extraction, on each suspicious cluster. Since mined flows are narrowly specified by restricted value ranges along *multiple* dimensions, this can give pinpoint identification of anomalous flows, in contrast to methods which examine fixed, single attribute fields (e.g. just looking at the range of destination IPs [4]).

An important distinction between techniques is whether they capture the hierarchical nature of attributes (and attribute conjunctions) in defining flows, e.g. [3] does while [18] does not. Generally there are two motivations for using a hierarchy. First, nodes at different levels capture data at different “scales”. Second, one may only care about the description at the leaves, where the high-volume flows are as precisely specified as possible. Even so, building a hierarchy may be the most efficient way of *computing* the leaf description. For purposes of this paper, this latter case

```

Begin {multidimensional clustering}
for  $k = 3$  to maxdepth
  for each significant node at level  $k - 1$ :
    1. Identify all of the node’s children (at level  $k$ ).
    2. for each child, if its flow size has not already been checked:
      i. Check whether all the 1-D ancestors of the flow are significant.
      ii. If i. is true, check whether all parents of the flow are significant.
      iii. If i. and ii. are true, measure the flow size using the Netflow table, as previously discussed. If the flow is above the threshold, save it (and its size) as a “significant” node in the hierarchy.
    end for
  end for
End {multidimensional clustering}

```

Table 1. The algorithm pseudocode for multidimensional clustering.

applies, with our trees defined as in [3][16]. A flow is “significant” if its aggregate traffic, measured over a given time interval, is greater than a threshold. A key property of the tree is that a child node can only be significant if *all* of its parent nodes are significant¹. This “significance property of the hierarchy” allows one, via a top-down, root-to-leaf process, to promptly *reject* many flows (those with any insignificant ancestors) and to identify the most specific significant flows in an efficient manner. In [3], a tree structure was advanced that captures the hierarchical nature both of individual attributes (source and destination IPs) and of collections of attributes. This hierarchy defines flows with flexible specificity. While [3] did aim to optimize mining efficiency, their work was proposed for off-line auditing. In [16], we significantly improved computational efficiency over [3], seeking viability as an on-line procedure, and made initial progress in applying flow mining to anomaly detection.

In [16], we considered flows defined by the 5-tuple (source IP, destination IP, source port, destination port, protocol). The IP addresses are hierarchical – binary prefixes of variable specificity, from 8 to 32 bits. The protocol was treated as flat, i.e., nonhierarchical. The ports were represented using a simple hierarchy, with the range divided into low (< 1024) and high (> 1023), and a flat representation for each sub-range. Each attribute may also take on a *wild card* (*), indicating this attribute is *not* used in defining the flow. The set of all flows, consistent with the above, is represented by a *multidimensional flow tree* [3]. The objective in [3] was to identify all “significant” nodes in the tree, with the count measured in packets or bytes.

The method [3] starts with the 1-D hierarchy for each attribute taken alone, identifying all “significant” flows for the given attribute. While 1-D clustering has digesting value on its own, its

¹ Except for the layer directly below the root of the tree (which contains *all* the traffic), nodes in the trees defined in [3][16] have *multiple* parents.

Figure 2. AIDE and unexpectedness distribution of Slammer worm trace.[16]

primary role is in speeding up the (subsequent) algorithm for building the multi-D hierarchy. This latter tree is built top-down, level-by-level, starting from the root ($k = 1$). In going from level $k - 1$ to level k , one is further specifying a single attribute value. The root node is always significant. The next level consists of (partitioned) flows along each attribute dimension. Each flow at this level is checked for significance. This is done by a flow matching operation, i.e. by matching the flow's definition against the definition of each flow in the Netflow table and, if a match occurs, adding to the current accumulated flow count. For each subsequent level, [3] identifies significant nodes via the algorithm in Table 1. To reduce the number of (computationally heavy) flow count passes taken over the Netflow table, the algorithm tests two necessary conditions for flow "significance" before actually measuring the size of a flow. Both tests reject flows that have insignificant ancestors.

Since a large fraction of the computation in [3] was spent in flow matching, we proposed paradigms for greatly reducing flow matches [16]. This was achieved capitalizing on the hierarchical structure, which allows measuring a flow's size by matching in a *small* table whose elements match the definition of a *parent* flow, rather than matching in the full (large) Netflow table. In experiments on the New Zealand (NZIX) trace data [15] reported in [16], our method reduced total execution time by a factor of 8.21, compared with [3]. For a 1-hour trace and a 5% threshold on flow size, execution on a Pentium-4 was less than 13 seconds.

We also evaluated the potential for identifying the anomalous clusters (flows), within the cluster pool produced by our mining – note that this pool in general contains *many* innocuous clusters, which should not be identified as anomalous. We proposed a simple detection criterion based on the (cluster-conditional) distributions of the IP header fields. Many criteria have been proposed in the past, e.g. monitoring ports, the distribution of destination IPs, and flow sizes. In [3], a criterion dubbed "unexpectedness" was proposed, based on the discrepancy between the actual flow volume and the volume predicted based on assuming statistically independent attributes. In [16], we proposed an alternative measure motivated by information theory, based on the (Shannon) entropy of individual header fields. For a given cluster, k , the source/destination IP entropy is:

$$H_k(IP) = -\sum_l P[l] \log P[l]$$

where l are IP addresses in cluster k that occurred in the current digest interval, and

$$P[l] = \frac{\text{number of packets (bytes) in cluster } k \text{ with IP } = l}{\text{number of packets (bytes) in cluster } k}$$

For packet (byte)-based reports, the probabilities $P[l]$ are computed based on packets (bytes). Entropy measures the degree of uncertainty in the IP address, conditioned on the packet (byte) belonging to the given cluster. For DDoS or flash crowd clusters, we would expect source IP entropy to be much larger than destination IP entropy. On the other hand, for a fast scanning worm, worm clusters should have larger destination IP entropy. We thus combined these entropies, forming a scalar statistic dubbed the absolute IP difference in entropy (AIDE), i.e.

$$AIDE(k) = |H_k(\text{source IP}) - H_k(\text{destination IP})|.$$

While it is possible to more generally look at entropies in multiple individual header fields, this simple (scalar) criterion is quite attack-discriminating, as was demonstrated for Slammer, DARPA, and CodeRed traces in [16], whereas "unexpectedness" in [3] was not found to be discriminating. An example, for the Slammer trace, is shown in Figure 2, with the top four AIDE-ranking clusters representing pure attack/worm flows. By contrast, "unexpectedness" does not help to identify the attack flows.

Directly Mining Suspicious Clusters

For digesting purposes, one may be interested in *all* (narrowly specified) significant flows. However, for detection one really only needs to identify the *anomalous* flows. Thus rather than first identifying all significant flows, it should be substantially more efficient to somehow only *directly* mine the (small) set of suspicious flows. A simple step along these lines is achieved by treating the destination port as a flat variable. This is reasonable since most packets from a common attack use the same port. This restriction not only removes a significant portion of the tree, but also helps to achieve greater *purity* of the attack within individual clusters. Purity is also enhanced by only considering significant *leaf* clusters (those without any significant descendants in the tree) – this is the most narrowly specified set of significant clusters. A more ambitious approach for improving efficiency is as follows: instead of solely using volume as the criterion for growing the tree, we could in principle *also* use IP entropies, creating child nodes only if they have both significant volume and IP entropy (source *or* destination) above a threshold. Unfortunately, entropy is not guaranteed to monotonically decrease when splitting a parent into its children. Thus, we instead propose to use the source and destination IP *cardinalities* of a cluster (which gives a (uniform model) approximation to the entropy). With thresholds properly set, low cardinalities (below threshold) may not *only* accurately identify a given node is "innocuous" – given that cardinalities *do* decrease as one descends the tree, it should also reliably indicate all the node's *descendants* are innocuous. In this case, we can prune this entire branch of the tree. Based on our experiments, this approach greatly reduces computations and memory while preserving (at least) the anomalous/attack clusters in the mined tree. After mining based on these two (cardinality and volume) criteria, we may additionally use AIDE or other criteria to more definitively identify the suspicious subset of clusters.

Figure 3. Suffix tree of input string “aaab”.

3.3 Worm Signature Extraction

A suffix tree [17][1], which has found wide use in general-purpose string matching, is a trie-like data structure that compactly represents a string. As shown in Figure 3, each suffix of the string “aaab” is represented as a leaf in the suffix tree, and the edges of the suffix tree are labeled with characters. The concatenation of the labeled edges on the path from the root to a given node, which is called the prefix of the node, represents a subset of the input string. An efficient algorithm has been proposed in [9] to construct a suffix tree in time and space linear in the length of the input string.

The suffix tree representation can be extended to deal with multiple input strings, with the result called a generalized suffix tree (GST). Figure 4 is an example of a GST of two input strings “aaab” and “aabb”. Just as for the simple suffix tree in Figure 3, every suffix in each string is represented as a leaf in a GST. We note that two or more input strings may have common suffixes, so it is possible that two or more leaves in a GST represent the same substrings. For example, “b” is a suffix of both “aaab” and “aabb”, so there are two leaves of “b” in the GST in Figure 4.

By adding a parent pointer for each node, both the prefix length and the occurrence counts of each node can be accumulated during the process of building a GST. Also, using a GST, instead of the CSS method [5] suggested in [11], we can find suspicious signatures in about half the time and space.

For each of the suspicious clusters obtained via our mining and anomaly detection process, a generalized suffix tree is built on the payloads of all packets that match the cluster definition. After we finish building the GST, we traverse all nodes in the tree and pick out the nodes which satisfy either of the following two criteria: 1) prefix longer than 40 bytes and frequency of occurrence greater than one percent of the total packets in the cluster; 2) prefix larger than 10 bytes and frequency larger than fifty percent of the total packets in the cluster. The first criterion aims at catching relatively long strings (or even whole payloads) that appear several times within the time interval, for a given cluster’s flow definition; the second is aimed at catching short strings which are nearly always present in different instantiations of the worm. Based on these two criteria, the prefix of each selected node is taken as a putative worm signature. It is noted that the length

Figure 4. Generalized suffix tree of input strings “aaab” and “aabb”.

threshold in the first criterion is the same as the window size used in Earlybird [14]. Because [14] claims that Earlybird can successfully extract signatures of more than ten popular worms such as Slammer, Blaster, etc., all of these worm signatures are also expected to be caught by our method, using the first criterion. Moreover, as discussed previously, the purity of the suspicious clusters from our mining is generally quite high, based on its (flat) treatment of the destination port and protocols and by restricting to considering leaf clusters. With highly pure suspicious clusters, the frequency threshold of fifty percent in the second criterion is, we believe, reasonable and acceptable.

However, there is great redundancy in the worm signatures selected based on the above two criteria. For example, if a 100-byte long string S satisfies the frequency threshold in the first criterion, any substring of S which contains more than 40 bytes also satisfies the first criterion. Thus, in order to identify a set of signatures that is both as specific and non-redundant as possible, we remove signatures that are substrings of other signatures and which appear with nearly the same frequency.

We also note that a false positive rate for each extracted signature can be measured with respect to an innocuous flow pool. We may, for example, have a legacy database that is representative of “normal” traffic, that can be used to form this pool. Alternatively, an innocuous pool can be derived during the mining process, from the set of flows that are *not* deemed suspicious (as next done in our results). We believe more accurate signatures can be extracted by jointly considering signature lengths, occurrence counts, and false alarm rate. One advantage of our signature extraction over that in Earlybird and Polygraph, based on our use of GST, is that we can easily modify our rules to search for multiple signature “descriptions” of varying length range, with little increase in complexity.

***** NETWORK INTRUSION DTECTION REPORT *****							
The network traffic begins at Jun 7, 2004 03:14:39 EST. The traffic length is 1.0 minutes.							
The total traffic is 78.5k packets (73.4M bytes). The threshold is 1% = 0.8k packets.							
No.	Src IP	Dst IP	SrcPt	DstPt	Prot	PktNo	Perc
1	192.168.114.97/32	0.0.0.0/1	high	110	6	1.1k	1.4%
Total: 21 signatures							
1)	Leng: 15	Freq: 898	False Pos.: 0				
2)	Leng: 100	Freq: 54	False Pos.: 0				
3)	Leng: 100	Freq: 36	False Pos.: 0				
4)	Leng: 100	Freq: 41	False Pos.: 0				
5)	Leng: 100	Freq: 46	False Pos.: 0				
6)	Leng: 100	Freq: 39	False Pos.: 0				
7)	Leng: 100	Freq: 39	False Pos.: 0				
8)	Leng: 100	Freq: 50	False Pos.: 0				
9)	Leng: 100	Freq: 46	False Pos.: 0				
10)	Leng: 100	Freq: 49	False Pos.: 0				
11)	Leng: 100	Freq: 46	False Pos.: 0				
12)	Leng: 100	Freq: 38	False Pos.: 0				
13)	Leng: 100	Freq: 42	False Pos.: 0				
14)	Leng: 100	Freq: 49	False Pos.: 0				
15)	Leng: 100	Freq: 39	False Pos.: 0				
16)	Leng: 100	Freq: 45	False Pos.: 0				
17)	Leng: 100	Freq: 44	False Pos.: 0				
18)	Leng: 100	Freq: 54	False Pos.: 0				
19)	Leng: 100	Freq: 38	False Pos.: 0				
20)	Leng: 100	Freq: 46	False Pos.: 0				
21)	Leng: 100	Freq: 57	False Pos.: 0				
2	192.168.114.97/32	0.0.0.0/1	high	4662	6	1.1k	1.4%
Total: 2 signatures							
1)	Leng: 101	Freq: 66	False Pos.: 27				
2)	Leng: 68	Freq: 18	False Pos.: 4				

Table 2. The mining and signature report of worm salted Taiwan trace.

4. EXPERIMENTAL METHODOLOGY

4.1 Polymorphism via Encryption Schemes

For polymorphic worms that scan expeditiously, we will initially assume in our experiments that each infective will either cycle

through a fixed and relatively small number of completely different malware cipher images pre-computed with a different kind of encryption for each, or a different malware cipher per scan (e.g., using a “key” dependent on the destination address); but for this latter case, common decryption code in plain text will be present (and “prevalent”) in all payloads (in addition to the

explicit code, of course [2]). That is, we assume that in one of these two ways content will be prevalent in the worm's packet-payloads. Other kinds of payload obfuscation are both clearly possible and more challenging to contend with – e.g., those that shuffle subroutines and branch around junk code at the binary level (many other source-level techniques exist). However a payload containing 1386 executable code of sufficient length can be accurately detected based on a known histogram of commands for this language (just as a one-to-one letter substitution cipher is broken). Moreover, the computation and time required to significantly obfuscate binary executable in this manner may slow down the progress of the worm and make it more noticeable to host-based defenses. Obviously, obfuscation techniques that significantly increase the amount of malware that needs to be transmitted, i.e., the number of bytes or packets, will also slow down the progress of the worm. The use of encryption also has these problems from the attacker's point-of-view. This tradeoff thus justifies the simple encryption-based polymorphism mechanisms that we consider here.

4.2 Issues in Salting Background with Worm Traffic

From the discussion of the previous subsection, we can argue that, for experimental purposes, the worm packet-payload's prevalent content can be generated at random. That said, for a *given* background trace ostensibly consisting of innocuous traffic, the prevalent content of the added worm packets may either be similar to or *very different from* innocuous payloads. Though we used randomly generated worm payloads in our experimental analysis subsequently described, one can easily envision an experimental process wherein the correlation between the worm packet-payload prevalent content and the innocuous payloads is varied. It would naturally be expected that an increase in this correlation would reduce the performance gains of payload inspection (in terms of false positives and/or missed detections) beyond that of standard intrusion detection based on TCP/IP header information alone.

Finally, one can also vary the volume and placement of malware within a trace. In a similar way, decreasing the volume of malware packets (relative to innocuous packets) may result in more missed detections.

5. PRELIMINARY EXPERIMENTAL RESULTS AND DISCUSSION

The original Taiwan trace was broken into one-minute segments, with mining and signature extraction performed on each. Polymorphic TCP worm packets, with some bytes common to every packet, were inserted into each segment, at different rates. We did not insert the SYN and SYN ACK packets of this TCP worm, only working on its polymorphic images at this time. We tried our mining, detection, and signature extraction processes on every segment, and the results for most segments are quite similar. Here we just describe the results obtained for a single one-minute segment. There were a total of 79,435 normal packets and the TCP worm was inserted at a rate of 15 packets per second, with, thus, a total of 898 worm packets added. The worm packets were randomly selected from 20 different images, each 100 bytes long and sharing 15 common bytes randomly positioned anywhere within the 100-byte payload. Here we assume the situation where

one computer inside the LAN is infected and pumps out worm packets, each with the same destination port number, to randomly selected destination IP addresses. Thus, the worm packets all use the same source IP address, destination port number, and protocol, but randomly generated destination IPs and source ports. In order to provide a reasonable test to the efficiency and accuracy of our proposed signature extraction method, we used destination port 110 and protocol 6 for the worm packets – there are normal flows specified by this pair which occur within the segment. As a result, some normal packets should be contained in the “worm clusters” our flow mining extracts, making these clusters somewhat impure and, thus, making signature extraction more difficult.

The mining and signature report is shown in Table 2², with ‘Leng’, ‘Freq’, and ‘False Pos.’ denoting the length, frequency, and false positive rate of a signature, respectively. In the signature extraction step, we collected all payloads of packets that match a cluster definition and concatenated them into a long string, on which a GST is built. In our experiments, we chose a (relatively low) volume threshold of 1% and an IP cardinality threshold of 3 (an appropriate choice for this particular trace). Even with these (relatively low) thresholds, there were only two suspicious significant leaf clusters found by our mining and anomaly detection processing. As an aside, we also note that mining to directly detect suspicious clusters, instead of just high volume clusters, was found to reduce computational complexity, compared to the method in [16], by a factor of roughly three for this (quite short) trace.

The first extracted cluster is a worm cluster that contains 898 worm packets and 245 normal packets. There were 21 signatures extracted from this cluster, *all* of them worm-related signatures, i.e., no signatures were extracted from the 245 normal packets in this cluster. The first 15-byte signature corresponds to the common bytes included in each worm image, and its frequency ‘898’ is exactly the number of worm packets inserted into the segment. The other 20 signatures each correspond to one of the 21 different worm images, with each signature 100 bytes long (the full image payload) and with the sum of their frequencies equaling the total number of worm packets (898). Thus, all the worm signatures, both common bytes and worm images, were correctly extracted and no false signatures were extracted, even though there were 20% normal packets in this cluster. Finally, when each of these 21 signatures was applied to all the traffic payloads in the segment *excluding* the suspicious clusters (this “complement” set forms our estimate of an innocuous flow pool), there were *zero* false alarms³.

The second cluster exclusively consists of packets from the original trace (before worm salting). Because there is peer-to-peer file transfer activity in the background traces, it is not surprising our mining and anomaly detection captures this activity in a suspicious cluster since peer-to-peer activity involves address dispersion that is also characteristic of a scanning worm. Moreover, the two relatively long signatures extracted from this cluster are also acceptable, considering that there may be common content in the packets that are peer-to-peer file transferred. It is possible to reject these signatures by monitoring whether or not

² We anonymized the source IP address in the table for privacy purpose.

³ The random generation of worm payloads may be a contributing factor to this result.

they recur over multiple mining intervals (we would expect worm signatures to have a higher recurrence rate than peer-to-peer activity), or by checking the content represented by these signatures. However, in this case, these signatures can also be correctly rejected based on the fact that each of them had a *positive* false alarm count, when measured against the (estimated) innocuous flow pool. In particular, these two signatures occurred 27 and 4 times, respectively, in packets with either a different destination port number or a different protocol from the definitions specified by the cluster⁴.

6. CONCLUSIONS

Based in part on our previous multidimensional flow mining and anomaly detection approach, a new payload-based IDS was introduced in this paper. Moreover, a new method, which focuses on extracting signatures from polymorphic worms, was developed by building a suffix tree for each suspicious cluster. The experimental results for the worm-salted Taiwan trace showed that our proposed method extracts variable-length worm signatures with high accuracy and low false alarms. In future work, we will investigate accuracy and false alarms on the length of the common bytes in the worm payloads. We will also explore use of polymorphic engine in generating actual worm binaries. Finally, to facilitate further development and verification of our results by third parties, all of our intrusion detection code and the Taiwan trace, with payload fingerprints, will be made available (via open-source license for the code) at emist.ist.psu.edu and www.isi.edu/deter.

7. ACKNOWLEDGMENTS

We wish to thank Felix Wu for supplying us with the Taiwan trace that we used for our experimental analysis.

8. REFERENCES

- [1] A. Apostolico, C. Iliopoulos, G.M. Landau, B. Schieber, and U. Vishkin. Parallel construction of a suffix tree with applications, *Algorithmica*, 1988.
- [2] J.R. Crandall, Z. Su, S.F. Wu, and F.T. Chong. On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits. In *ACM Conf. Computer and Communication Security*, 2005.
- [3] C. Estan, S. Savage, and G. Varghese. Automatically inferring patterns of resource consumption in network traffic. In *Proc. ACM SIGCOMM*, 2003.
- [4] L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to DDoS attack detection and response. In *Proc. DARPA Info. Survivability Conf. and Expo.*, 2003.
- [5] L. Hui. Color set size problem with applications to string matching. In *Proc. Symp. Comb. Pattern Matching*, 1992.
- [6] H.A. Kim and B. Karp. Autograph: toward automated, distributed worm signature detection. In *Proc. USENIX Security Symp.*, 2004.
- [7] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. In *Proc. ACM SIGCOMM*, 2005.
- [8] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. Intel. Conf. Very Large Databases*, 2002.
- [9] E.M. McCreight. A space-economical suffix tree construction algorithm. In *Journal of the ACM*, 23(2):262--272, 1976.
- [10] D. Moore, C. Shannon, G. Voelker, and S. Savage. Internet quarantine: requirements for containing self-propagation code. In *IEEE Proc. INFOCOM*, 2003.
- [11] J. Newsome, B. Karp, and D. Song. Polygraph: automatically generating signatures for polymorphic worms. In *Proc. IEEE Symp. Security and Privacy*, 2005.
- [12] M.O. Rabin. Fingerprinting by random polynomials, Technical Report TR-15-81, Harvard University, 1981.
- [13] W. Scheirer and M. Chuah. The strength of syntax based approaches to dynamic network intrusion detection. In *Proc. Conf. on Info. Sci. & Sys.*, Princeton, NJ, Mar. 2006.
- [14] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proc. ACM/USNIX Symp. Operating System Design and Implementation*, 2004.
- [15] Waikato Applied Network Dynamics Research Group. Auckland University data traces. <http://wand.cs.waikato.ac.nz/wand/wits/>.
- [16] J. Wang, D.J. Miller, and G. Kesidis. Efficient Mining of the Multidimensional Traffic Cluster Hierarchy for Digesting, Visualization, and Anomaly Identification. To appear in IEEE JSAC on High-Speed Network Security, 2005.
- [17] P. Weiner. Linear pattern matching algorithms. In *Proc. 14th IEEE Symp. Switching and Automata Theory*, 1973.
- [18] K. Xu, Z. Zhang and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. In *Proc. ACM SIGCOMM*, 2005.
- [19] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund. Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications. In *Proc. Internet Meas. Conf.*, 2004.

⁴ We measured a false alarm count, rather than rate, due to the relatively small number of packets in the segment.