Traffic Aggregation for Malware Detection

Ting-Fang Yen[1] and Michael K. Reiter[2]

[1] Carnegie Mellon University
tingfang@cmu.edu
[2] University of North Carolina, Chapel Hill
reiter@cs.unc.edu

Abstract. Stealthy malware, such as botnets and spyware, are hard to detect because their activities are subtle and do not disrupt the network, in contrast to DoS attacks and aggressive worms. Stealthy malware, however, does communicate to exfiltrate data to the attacker, to receive the attacker's commands, or to carry out those commands. Moreover, since malware rarely infiltrates only a single host in a large enterprise, these communications should emerge from multiple hosts within coarse temporal proximity to one another. In this paper, we describe a system called TĀMD (pronounced "tamed") with which an enterprise can identify candidate groups of infected computers within its network. TĀMD accomplishes this by finding new communication "aggregates" involving multiple internal hosts, i.e., communication flows that share common characteristics. We describe characteristics for defining aggregates—including flows that communicate with the same external network, that share similar payload, and/or that involve internal hosts with similar software platforms—and justify their use in finding infected hosts. We also detail efficient algorithms employed by TĀMD for identifying such aggregates, and demonstrate a particular configuration of TĀMD that identifies new infections for multiple bot and spyware examples, within traces of traffic recorded at the edge of a university network. This is achieved even when the number of infected hosts comprise only about 0.0097% of all internal hosts in the network.

1 Introduction

It is clearly in the interest of network administrators to detect computers within their networks that are infiltrated by spyware or bots. Such stealthy malware can exfiltrate sensitive data to adversaries, or lie in wait for commands from a bot-master to forward spam or launch denial-of-service attacks, for example. Unfortunately it is difficult to detect such malware, since by default it does little to arouse suspicion: e.g., generally its communications neither consume significant bandwidth nor involve a large number of targets. While this changes if the bots are enlisted in aggressive scanning for other vulnerable hosts or in denial-of-service attacks—in which case they can easily be detected using known techniques (e.g., [38, 27])—it would be better to detect the bots prior to such a disruptive event, in the hopes of averting it. Moreover, such easily detectable behaviors are uncharacteristic of significant classes of malware, notably spyware.

We hypothesize that even stealthy, previously unseen malware is likely to exhibit communication that is detectable, if viewed in the right light. First, since emerging malware rarely infects only a single victim, we expect its characteristic communications, however subtle, to appear roughly coincidentally at multiple hosts in a large network.

D. Zamboni (Ed.): DIMVA 2008, LNCS 5137, pp. 207–227, 2008.

Second, we expect these communications to share certain features that differentiate them from other communications typical of that network. Of course, these two observations may pertain equally well to a variety of communications that are not induced by malware, and consequently the challenge is to refine these observations so as to be useful for detecting malware in an operational system.

In this paper we describe such a system, called TĀMD, an abbreviation for "Traffic Aggregation for Malware Detection". As its name suggests, TĀMD distills *traffic aggregates* from the traffic passing the edge of a network, where each aggregate is defined by certain characteristics that the traffic grouped within it shares in common. By refining these aggregates to include only traffic that shares multiple relevant characteristics, and by using past traffic as precedent to justify discarding certain aggregates as normal, TĀMD constructs a small set of new aggregates (i.e., without previous precedent) that it recommends for examination, for example, by more targeted (e.g., signature-based) intrusion detection tools. The key to maximizing the data-reducing precision of TĀMD is the characteristics on which it aggregates traffic, which include:

- **Common destinations:** TĀMD analyzes the networks with which internal hosts communicate, in order to identify aggregates of communication to busier-than-normal external destinations. Spyware reporting to the attacker's site or bot communication to a bot-master (e.g., with IRC, HTTP, or another protocol) might thus form an aggregate under this classification.
- **Similar payload:** TĀMD identifies traffic with similar payloads or, more specifically, payloads for which a type of edit distance (*string edit distance matching with moves* [8]) is small. Intuitively, command-and-control traffic between a bot-master and his bots should share significant structure and hence, we expect, would have a low edit distance between them.
- **Common internal-host platforms:** TĀMD passively fingerprints platforms of internal hosts, and forms aggregates of traffic involving internal hosts that share a common platform. Traffic caused by malware infections that are platform-dependent should form an aggregate by use of this characteristic.

Alone, each of these methods of forming traffic aggregates would be far too coarse to be an effective data-reduction technique for identifying malware, as legitimate traffic can form aggregates under these characterizations, as well. In combination, however, they can be quite powerful at extracting aggregates of malware communications (and relatively few others). To demonstrate this, we detail a particular configuration of TĀMD that employs these aggregation techniques to identify internal hosts infected by malware that reports to a controller site external to the network. Indeed, botnets have been observed to switch controllers or download updates frequently, as often as every two or three days [19, 11]; each such event gives TĀMD an opportunity to identify these communications. We show that with traffic generated from real spyware and bot instances, TĀMD was able to reliably extract this traffic from all traffic passing the edge of a university network, while the number of other aggregates reported is very low.

In addition to identifying aggregates and ways of combining them to find malware-infected hosts, the contributions of TĀMD include algorithms for computing these aggregates efficiently. Our algorithms draw from diverse areas including signal processing, data mining and metric embeddings. We will detail each of these algorithms here.

2 Related Work

Botnet detection. Previous approaches to botnet detection rely on heuristics that assume certain models of botnet architecture or behavior, such as IRC-based command-and-control [7, 4, 26, 11], the presence of scanning activities, long idle time and short response time for bots compared to humans [32], etc. Karasaridis et al. [19] proposed an approach for identifying botnet controllers by combining heuristics that assume the use of IRC communication, scanning behavior, and known models of botnet communication. BotHunter [14] models all bots as sharing common infection steps—namely target scanning, infection exploit, binary download and execution, command-and-control channel establishment, and outbound scanning—and then employs Snort with various malware extensions to raise an alarm when a sufficient subset of these are detected. Thus, malware not conforming to this profile (e.g., spyware or bots engineered differently) would seemingly go undetected by their approach. Ramachandran et al. [36] observed that botmasters lookup DNS blacklists to tell whether their bots are blacklisted. They thus monitor lookups to a DNS-based blacklist to identify bots.

We believe our approach to be fundamentally different from the above approaches in the following respect. While these approaches work from models of malware behavior (not unlike signature-based intrusion detection), our approach simply seeks to identify new aggregates of communication that are not explained by past behavior on the network being monitored. Like all anomaly-detection approaches, our challenge is to demonstrate that the number of identified anomalous aggregates is manageable, but it has the potential to identify a wider range of as-yet-unseen malware. In particular, the assumptions underlying previous systems present opportunities for attackers to evade these systems by changing the behavior of botnets, and these systems will fail to detect other types of malware (e.g., spyware) that do not meet these assumptions.

Independently of or subsequently to our work [37], other works have begun to incorporate aspects of using aggregation for detecting bots. For example, BotSniffer [15] looks for infected hosts displaying spatial-temporal similarity. It identifies hosts with similar suspicious network activities, namely scanning and sending spam emails, and who also share common communication contents, defined by the number of shared bi-grams. BotMiner [13] groups together hosts based on destination or connection statistics (i.e., the byte count, the packet count, the number of flows, etc.), and on their suspected malicious activities (i.e., scanning, spamming, downloading binaries, or sending exploits). BotMiner is more similar to TĀMD in the sense that they both identify hosts sharing multiple common characteristics, but the characteristics on which TĀMD and BotMiner cluster hosts are different. BotSniffer seeks to identify known bot activities, such as scanning or spamming, and limits its attention only to bots using IRC or HTTP to communicate with a centralized botmaster.

Various prior works on botnet detection use honeypots (e.g., [2, 33]). As honeypots can only approximately mimic (at best) real user behavior, they may not attract spyware or bots that rely on human action to infect users' machines. Our approach, in not requiring a honeypot, places no assumptions about the infection vector by which attacks occur and whether these vectors present themselves in a honeypot. In doing so, we hope to make our approach as general as possible.

Techniques. The techniques we employ for aggregation, specifically on the basis of external subnets to which communication occurs, include some drawn from the signal processing domain. While others have drawn from this domain in the detection of network traffic anomalies, our approach has different goals and hence applies these techniques differently. Coarsely speaking, past approaches extract packet header information, such as the number of bytes or packets transferred for each flow, counts of TCP flags, etc., in search of volume anomalies like denial-of-service attacks, flash crowds, or network outages [39, 3, 22]. Lakhina et al. [25] studied the structure of network flows by decomposing OD flows (flows originating and exiting from the same ingress and egress points in the network) using Principal Component Analysis (PCA). They expressed each OD flow as a linear combination of smaller "eigenflows", which may belong to deterministic periodic trends, short-lived bursts, or noise, in the traffic. Terrell et al. [40] focused on multi-variate data analysis by grouping network traces into time-series data and selecting features of the traffic from each time bin, including the number of bytes, packets, flows, and the entropy of the packet size and port numbers. They applied Singular Value Decomposition (SVD) to the time-series data. From examining the low-order components, they were able to detect denial-of-service attacks. In general, transient and light-weight events would go unnoticed by these approaches, such as spammers that send only a few emails over the course of a few minutes [35]. Our work, on the other hand, is targeted at such lighter-weight events and so employs these techniques differently, not to mention techniques from other domains (e.g., metric embeddings, passive fingerprinting). Ramachandran et al. [34], in assuming that spammers exhibit similar email-sending behaviors across domains, constructed patterns corresponding to the amount of emails sent to each domain by known spammers. The patterns are calculated from the mean of the clusters generated through spectral clustering [6]. This is similar to our method of finding flows destined to the same external subnets; however, they do not look at other aspects of spamming besides the destination.

Another technique we employ is payload inspection, specifically to aggregate flows based on similar content. Payload inspection has been applied within methods for detecting worm outbreaks and generating signatures. Many previous approaches assume that malicious traffic is significantly more frequent or wide-spread than other traffic, and so the same content will be repeated in a large number of different packets or flows (e.g., [38, 21, 29, 18, 30]); we do not make this assumption here. Previous approaches to comparing payloads includes matching substrings [28, 18] or n-grams [42, 29, 15], hashing blocks of the payload [38, 22], or searching for the longest common substring [24]. Compared to these methods, our edit distance metric is more sensitive and accurate in cases where parts of the message are simply shifted or replaced. Goebel et al. [11] inspected packet payload to find IRC bots with formatted nicknames. They observed that often IRC bots have nicknames with common patterns, such as long random numbers or country codes. However, this approach can only detect bots for which the nickname format is known. ARAKIS from CERT Polska (`http://www.arakis.pl`) is an early-warning system that generates signatures for new threats. Assuming new attacks will have payloads not seen previously, they examine traffic from honeypots and darknets to cluster flows with similar content (determined by comparing Rabin hashes) not seen before, and that are performing similar activities, i.e., port scanning. A signature is

generated from the longest common substrings of the similar flows. However, ARAKIS currently only focuses on threats that propagate through port scanning.

Another tool for intrusion analysis is the commercial product StealthWatch from Lancope (`http://www.lancope.com`). StealthWatch monitors all traffic at the network border, checking for policy violations or signs of anomalous behavior by looking for higher-than-usual traffic volumes. Although this is similar to our approach of using past traffic as a baseline for identifying busier-than-normal external destinations, they does not refine this information using, e.g., payload or platform aggregation as we do here. Thus, it is primarily useful for detecting only large-volume anomalies like port scanning and denial-of-service attacks.

3 Defining Aggregates

Given a collection of bi-directional flow records observed at the edge of an enterprise network, our system aims to identify infected internal hosts by finding communication "aggregates", which consist of flows that share common network characteristics. Specifically, TAMD deploys three aggregation functions to identify flows with the following characteristics: those that contribute to busier-than-usual destinations, that have payloads for which a type of edit distance is small, or that involve internal hosts of a common platform.

The aggregation functions take as input collections of flow records, Λ, and output either groups (aggregates) of internal hosts that share particular properties or a value indicating the amount of similarity between the input flow record collections. We presume that each flow record $\lambda \in \Lambda$ includes the IP address of the internal host $\lambda.\mathsf{internal}$ involved in the communication and the external subnet $\lambda.\mathsf{external}$ with which it communicates. λ also includes some portion of the payload $\lambda.\mathsf{payload}$ of that communication, packet header fields, and the start and end time of the communication.

3.1 Destination Aggregates

Previous studies show that the destination addresses with which a group of hosts communicates exhibit stability over time, both in the amount of traffic sent and in the set-membership of the destinations [1, 23]. Malware activities are thus likely to exhibit communication patterns outside the norm, i.e., contacting destinations that the internal hosts would not have contacted otherwise.

The destination aggregation function $\mathsf{ByDest}^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$ takes as input two sets $\Lambda, \Lambda_{\mathsf{past}}$ of communication records. The variable τ is a parameter to the function, as described later in this section. By analyzing the external addresses with which internal hosts communicate in Λ and Λ_{past}, the function outputs a set $\mathsf{SuspiciousSubnets}$ of destination subnets for which there is a larger number of interactions with the internal network, using Λ_{past} as a baseline. The function also outputs an integer $\mathsf{numAggs}$ and a set Agg_i $(1 \leq i \leq \mathsf{numAggs})$, where Agg_i are internal hosts (IP addresses) that originated traffic in Λ, and who contributed to larger-than-usual number of interactions with an external destination subnet in $\mathsf{SuspiciousSubnets}$.

At a high level, the set SuspiciousSubnets of selected "suspicious" external destinations is determined after filtering out periodic and regular activities in the communications of the network as represented in the past traffic Λ_{past}. External destinations observed in Λ that do not follow the norm, i.e., that according to Λ_{past} are busier than usual or have not been contacted before, are thus output in SuspiciousSubnets.

Below we describe the three processing steps in $\mathsf{ByDest}^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$: (i) Trend filtering, which selects the set of suspicious external destinations; (ii) Dimension reduction, which first characterizes each host by a vector indicating which suspicious destinations it interacted with, and then reduces the dimensionality of these vectors while preserving most of the information; and (iii) Clustering, which forms clusters of the vectors (i.e., internal hosts) by the destinations they contacted.

Trend Filtering. Trend filtering aims to remove regular and periodic communications from Λ, so that external destinations showing behavior outside the norm are identified. In particular, the "norm" is defined, for each external destination subnet, by the average number of internal hosts that communicate with that subnet in various periodic intervals, as recorded in Λ_{past}. For example, periodic patterns, such as Windows machines connecting to the Windows update server on a weekly basis or banking websites experiencing traffic spikes on pay day each month, can be inferred from Λ_{past}. The change in activity of a destination in Λ can then be measured by how much more traffic it received in Λ compared to its average values for previous time intervals in Λ_{past}. In the current implementation, a destination is selected to be in SuspiciousSubnets if no internal host has been seen to communicate with it for all previous periodic time intervals in Λ_{past}.

Dimension Reduction. Given SuspiciousSubnets, each internal host can be represented as a binary vector $v = (v[1], v[2], \cdots, v[k])$ for which the dimensionality k is equal to the number of destinations in SuspiciousSubnets. A dimension $v[i]$ is set to 1 if the internal host communicated with destination i in SuspiciousSubnets (according to Λ), and 0 otherwise. However, the dimensions may be redundant or dependent on one another; e.g., retrieving a web page can cause other web servers to be contacted. To identify such relationships between the destinations and to further dimension reduction, we apply Principal Component Analysis (PCA).

PCA [17] is a method for analyzing multivariate data. It enables data reduction by transforming the original vectors onto a new set of orthogonal axes, i.e., principal components, while preserving most of the original information. This is done by having each principal component capture as much of the variability in the data as possible.

While a vector originally has length equal to the number of suspicious destinations, the transformed vector after PCA has a dimensionality that is the number of selected principal components, with each dimension now representing a linear combination of the external destinations. The number of selected principal components depends on the amount of variance we want to capture in the data, denoted as the parameter τ. The more variance to be captured, the more accurate the transformation represents the original data, but, at the same time, more principal components are needed, increasing the dimensionality.

Clustering. PCA reduces the vector dimensionality significantly, after which hosts connecting to the same combinations of destinations can be identified efficiently through clustering. $\mathsf{ByDest}^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$ forms clusters of the vectors (i.e., internal hosts) whose traffic is present in Λ using a K-means clustering algorithm [20], which does not require the number of clusters to be known in advance.

1. Randomly select a vector as the first cluster hub. Assign all vectors to this cluster.
2. Select the vector furthest away from its hub as a new cluster hub. Re-assign all vectors to the cluster whose hub it is closest to.
3. Repeat step 2 until no vector is further from its hub than half of the average hub-hub distance.

Cosine distance is used for comparing vector distances, i.e., $\mathsf{CosineDist}(v_1, v_2) = \cos^{-1}((v_1 \bullet v_2)/(|v_1||v_2|))$, for two vectors v_1 and v_2, where the symbol $\bullet$ is the dot product between the two vectors, and $|v_1|$ is the length of vector v_1. Cosine distance is essentially a normalized dot product of the vectors, where a particular dimension would contribute to the final sum if and only if both vectors have a nonzero value in that dimension. In our case, each vector represents a particular internal source host, and each dimension represents a linear combination of destination subnets. Cosine distance thus captures well the relationship between source hosts based on the common destinations they contacted.

Let $\mathsf{numAggs}$ denote the number of clusters from the above algorithm, and let Agg_i ($i = 1 \ldots \mathsf{numAggs}$) denote the hosts whose vectors comprise the i-th cluster. As such, Agg_i is an aggregate of internal hosts interacting with the same busier-than-usual external subnets. Again, all of $\mathsf{SuspiciousSubnets}$, $\mathsf{numAggs}$ and $\{\mathsf{Agg}_i\}_{1 \le i \le \mathsf{numAggs}}$ are output from $\mathsf{ByDest}^{\tau}(\Lambda,\Lambda_{\mathsf{past}})$.

3.2 Payload Aggregates

Payload inspection algorithms for malware detection have previously focused on either modeling byte-frequency distributions (e.g., [38, 21, 29, 18]), which assumes that malicious traffic should exhibit an observably different byte-frequency distribution from that of normal traffic, or substring matching (e.g., [42, 28]). In contrast to these approaches, our measure of payload similarity is *edit distance with substring moves*, which we choose because it is capable of capturing syntactic similarities between strings, even if parts of one string are simply shifted or replaced. To our knowledge, ours is the first work that detects malicious traffic by computing (a type of) string edit distance between payloads, and that develops techniques to scale these computations to high data rate environments.

For two character strings s_1 and s_2, $\mathsf{EditDist}(s_1, s_2)$ is defined as the number of character insertions, deletions, substitutions, or substring moves, required to turn s_1 into s_2. Given a string $s = s[1] \cdots s[\mathsf{len}(s)]$, a substring move with parameters i, j, and k transforms s into $s[1] \cdots s[i-1], s[j] \cdots s[k-1], s[i] \cdots s[j-1], s[k] \cdots s[\mathsf{len}(s)]$ for some $1 \le i \le j \le k \le \mathsf{len}(s)$. For example, swapping labeled parameters in a parameter list would be a substring move in a command string.

The payload comparison function $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}(\Lambda)$ that we introduce for use in Section 4 takes as input a set Λ of communication records, and outputs a value in the

range $[0, 1]$. It is parameterized by an edit distance threshold δ_{Ed} that determines if communication records λ, λ' are "close enough", i.e., if $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$. Its output indicates from among all pairs $(\lambda, \lambda') \in \Lambda \times \Lambda$ such that $\lambda.\mathsf{external} = \lambda'.\mathsf{external}$ (i.e., that involve the same external subnet) and $\lambda.\mathsf{internal} \neq \lambda'.\mathsf{internal}$ (i.e., that are not from the same internal host), the (approximate, see below) fraction for which $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$.

Since Λ can be large, computing $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}(\Lambda)$ by computing $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload})$ for each relevant (λ, λ') pair individually can be prohibitively expensive, i.e., requiring time proportional to $|\Lambda| \cdot |\Lambda|$, where $|\Lambda|$ denotes the cardinality of Λ. A contribution of our work is an algorithm for approximating the fraction of relevant record pairs (λ, λ') that satisfy $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$ in time roughly proportional to $|\Lambda|$ if δ_{Ed} is small.

To perform this approximation, we first *embed* the EditDist metric within L1 distance L1Dist, where for two vectors $v_1 = v_1[1 \ldots m]$, $v_2 = v_2[1 \ldots m]$, $\mathsf{L1Dist}(v_1, v_2) = \sum_{i=1}^{m} |v_1[i] - v_2[i]|$. That is, we transform each $\lambda.\mathsf{payload}$ into a vector v_λ so that if $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$ then $\mathsf{L1Dist}(v_\lambda, v_{\lambda'}) \leq \delta_{\mathsf{L1}}$ for a known value δ_{L1}. We do so using an algorithm due to Cormode et al. [8] called *Edit Sensitive Parsing* (ESP). For this algorithm, the ratio of δ_{L1} over δ_{Ed} is bounded by $O(\log n \log^* n)$, where n is the length of $\lambda.\mathsf{payload}$.[1] In our evaluation in Section 5, $n = 64$ and we set $\delta_{\mathsf{L1}} = \delta_{\mathsf{Ed}} \cdot \log_{10} 64$.

The embedding of EditDist into L1Dist is essential to our efficiency gains, since it enables us to utilize an approximate nearest-neighbor algorithm called *Locality Sensitive Hashing* (LSH) [10] to find vectors (and hence payload strings) near one another in terms of L1Dist (and hence in terms of EditDist), in time roughly proportional to $|\Lambda|$. Briefly, LSH hashes each vector using several randomly selected hash functions; each hash function maps the vector to a *bucket*. LSH ensures that if $\mathsf{L1Dist}(v_1, v_2) \leq \delta_{\mathsf{L1}}$, then the buckets to which v_1 and v_2 are hashed will overlap with high probability (and will overlap with much lower probability if not), where probabilities are taken with respect to the random selection of the hash functions. Consequently, we hash v_λ for each $\lambda \in \Lambda$, and explicitly confirm that $\mathsf{EditDist}(\lambda.\mathsf{payload}, \lambda'.\mathsf{payload}) \leq \delta_{\mathsf{Ed}}$ only for pairs (λ, λ') for which v_λ and $v_{\lambda'}$ hash to at least one overlapping bucket.

While edit distance may not be meaningful for encrypted messages, we can generalize the payload comparison function to define encrypted payload (e.g., detected by its entropy) as "similar". Exploring payload aggregation using other metrics is part of ongoing work; see Section 6.

3.3 Platform Aggregates

Forming traffic aggregates based on platform can be useful in identifying malware infections that are platform dependent. That is, suspicious traffic common to a collection of hosts becomes even more suspicious if the hosts share a common software platform.

Much host platform information can be inferred from traffic observed passively. Passive tools, unlike active fingerprinting tools like Nmap (`http://insecure.org`),

[1] $\log^* n$ denotes the *iterated logarithm* of n, i.e., the number of times the logarithm must be iteratively applied before the result is less than or equal to one.

do not probe hosts, but rather listen silently. The most comprehensive passive operating system fingerprinting tool of which we are aware is p0f (`http://lcamtuf.coredump.cx/p0f.shtml`), which extracts various IP and TCP header fields from SYN packets and uses a rule-based comparison algorithm. However, p0f cannot be applied to traffic traces in the flow-record format available to us (see Section 5), since most individual packet information (including for SYN packets) is not retained.

At the time of this writing, TĀMD employs two heuristics for fingerprinting internal host operating systems passively. The first employs time-to-live (TTL) fields witnessed at the network border in packets from internal hosts. It is well-known that in many cases, different operating system types select different initial TTL values (e.g., see `http://secfr.nerim.net/docs/fingerprint/en/ttl_default.html`). With a detailed map of the internal network, the observed TTL values can be used to infer the exact initial TTL value and so narrow the possibilities for operating system the host is running. However, a detailed map is typically unnecessary, as routes in most enterprise networks are sufficiently short that witnessing TTLs of packets as they leave the network enables the initial TTL values to be inferred well enough.

The second heuristic employed in TĀMD watches for host communications characteristic of a particular operating system platform. For example, Windows machines connect to the Microsoft time server by default during system boot for time sychronization, and the FreeBSD packages FTP server is more likely to be accessed by FreeBSD machines to install software updates. Once characteristic communications for platforms are identified, TĀMD can monitor for these to learn the platform of an internal host.

There are at least three limitations of such passive fingerprinting approaches for our purposes. First, DHCP-assigned IP addresses can be assigned to hosts with different operating systems over time, leading to inconsistent indications of the host operating system associated with an IP address. This suggests that TĀMD should weigh recent indications more heavily than older (and hence potentially stale) indications. Second, a machine with a compromised kernel could, in theory, alter its behavior to masquerade as a different operating system. In the absence of a possible IP address reassignment (e.g., for address ranges not assigned via DHCP), such a shift in behavior should itself be detectable evidence that a compromise may have occurred. In general, however, this limitation is intrinsic to *any* fingerprinting technique, passive or active, except those based on attestations from trusted hardware (e.g., TCG's Trusted Platform Module, `https://www.trustedcomputinggroup.org/groups/tpm/`). While we are unaware of malware that employs such a masquerading strategy, should platform-based aggregation for malware detection become commonplace, such systems would presumably need to migrate to attestation-based platform identification as it matures, in order to detect kernel-level compromises. User-level compromise should not affect platform-based aggregation using conventional fingerprinting techniques, however. The third limitation to forming aggregates based on platform is that it is likely for an enterprise to have the majority of its hosts running the same operating system. Thus ByPlatform would be more effective for networks with a diverse host population; for example, in a university setting.

Presently TĀMD uses the aforementioned heuristics based on TTL values and communication with characteristic sites to identify platforms. For use in Section 4, we

embody this in a function $\mathsf{ByPlatform}(\Lambda)$ that returns the largest fraction of internal hosts in Λ (i.e., among the hosts $\{\lambda.\mathsf{internal} : \lambda \in \Lambda\}$) that can be identified as having the same operating system, based on these heuristics applied to the traffic records Λ.

4 Example Configuration

In this section, we detail a configuration of TĀMD that identifies internal hosts infected by malware by employing the functions described in Section 3. This configuration identifies platform-dependent malware infections that report to common sites, e.g., IRC channels for receiving commands, public servers for downloading binaries, denial-of-service victims to attack, or database servers for uploading stolen information. This configuration is based on several observations about such malware:

O1. For even moderately aggressive malware, it is rarely the case that only a single victim exists in a large enterprise network, and so we hypothesize that stealthy malware is likely to generate traffic that appears within the same, coarse window of time (e.g., within the same hour) from multiple infected hosts. Moreover, we would expect that the controller site is located in a subnet that would not be a common one with which benign hosts interact, as major services with substantial client populations are typically better managed. As such, malware interacting with the controller site should generate a noticeable increase in the number of interactions with the controller's subnet in that window of time.

O2. We expect that the multiple instances of the malware communication to the controller site would be syntactically similar to each other, since the malware instances are communicating using the same protocol, and likely to be receiving or responding to similar commands.

O3. In the case of platform-dependent malware, the malware communications to the controller site will involve internal hosts all having the same host platform.

Using these observations, we have assembled the aggregation functions described in Section 3 into an algorithm FindSuspiciousAggregates to identify such malware infections, shown in Figure 1. The input to this function is a set Λ of traffic records observed in a fixed time interval (e.g., one hour) at the border of the network, and a set Λ_{past} of records previously observed at the border of the network. FindSuspiciousAggregates assembles and returns (in line 108) a set SuspiciousAggregates comprised of suspicious aggregates, where each aggregate is a set of internal hosts (IP addresses) that is suspected of being infected by malware.

FindSuspiciousAggregates first exploits observation O1, using ByDest^{τ} from Section 3.1 to find suspicious external subnets SuspiciousSubnets responsible for noticeably greater communication with the monitored network than in the past, and to find aggregates $\{\mathsf{Agg}_i\}_{1 \le i \le \mathsf{numAggs}}$, each of which includes internal hosts that interacted with one or more of these subnets. In line with observation O2, each aggregate is tested in line 105 to determine if distinct hosts in the aggregate communicate with suspicious subnets using similar payload. Finally, as motivated by observation O3, for each aggregate that has survived these tests, the platforms of the hosts in the aggregate are

```
FindSuspiciousAggregates(Λ, Λ_past)
100: SuspiciousAggregates ← ∅
101: (SuspiciousSubnets, numAggs, {Agg_i}_{1≤i≤numAggs}) ← ByDest^τ(Λ, Λ_past)
                                        /* Form aggregates by external subnet */
102: for i = 1 ... numAggs do
103:     Λ_i ← {λ ∈ Λ : λ.internal ∈ Agg_i}          /* Traffic from hosts in Agg_i */
104:     Λ_i^susp ← {λ ∈ Λ_i : λ.external ∈ SuspiciousSubnets}
                                        /* Traffic from hosts in Agg_i to suspicious subnets */
105:         if ByPayload^{δ_Ed}(Λ_i^susp) > 0.3 then
                                        /* Keep if traffic to same external subnet is self-similar */
106:             if ByPlatform(Λ_i^susp) > 0.9 then
                                        /* Keep if most of aggregate consists of one platform */
107:                 SuspiciousAggregates ← SuspiciousAggregates ∪ {Agg_i}
108: return SuspiciousAggregates
```

Fig. 1. The function used to find suspicious aggregates in the example construction given in Section 4. ByDest^{τ} (line 101), $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$ (line 105), and ByPlatform (line 106) are defined in Sections 3.1, 3.2 and 3.3, respectively.

inferred using ByPlatform and, if the aggregate is adequately homogenous (line 106), then it is added to SuspiciousAggregates (line 107).

There are numerous constants in Figure 1 that we have chosen on the basis of our evaluation that we will present in Section 5. These constants include $\tau = 90\%$ or 95% for ByDest^{τ}, 0.3 in line 105 and 0.9 in line 106. In addition, as we will describe in Section 5, the data on which we perform our evaluation includes 64 bytes of payload per record λ, for which we found $\delta_{\mathsf{Ed}} = 15$ to be an effective value. However, we emphasize that all of these constants can be adjusted in order to make this configuration of TĀMD more conservative or liberal in its selection of suspicious aggregates, and we plan to continue evaluation of the alternatives in ongoing work. That said, in Section 5, we show that with traffic generated from real spyware and bot instances, and traces from real bots captured in a honeynet, this configuration of TĀMD was able to reliably extract malware traffic from all traffic passing the edge of a university network, while the number of other aggregates reported is very low. This reliability is achieved even in tests where the number of simulated infected hosts comprise only about 0.0097% of the total number of internal hosts in the network, calculated as the maximum number of internal IP addresses observed communicating in any one hour period during our data collection (see Section 5), which was over 33,000.

5 Evaluation

We present an evaluation of the particular configuration of TĀMD described in Section 4, using traffic from real spyware and bot instances, which are overlaid onto flow records recorded at the edge of a campus network. The performance of TĀMD as observed in this evaluation is described in Appendix C.

5.1 Data Collection

Our network traffic traces were obtained from the edge routers on the Carnegie Mellon University campus network, which consists of two /16 subnets. The packets are organized into bi-directional flow records by Argus (Audit Record Generation and Utilization System, `http://www.qosient.com/argus`), which is a real time flow monitor based on the RTFM flow model [5, 16]. Argus inspects each packet and groups together those with the same attribute values into one bi-directional record. In particular, TCP and UDP flows are identified by the 5-tuple (source IP address, destination IP address, source port, destination port, protocol)[2], and packets in both directions are recorded as a summary of the communication, namely, an Argus flow record.

The fields extracted from Argus records are listed in Table 1. The rate of the traffic from the edge of our campus network is about 5000 flow records per second. The traces were collected for three weeks in November and December 2007. In our evaluation, we focused on TCP and UDP traffic.

Table 1. Extracted Flow Fields

IP Header	Transport Header	Flow Attribute
Source IP	Source Port	Byte Count
Destination IP	Destination Port	Packet Count
Protocol	TCP Sequence Number	Payload (64 bytes)
TTL	TCP Window Size	

We also obtained network traffic traces for several malware. The malware traces used for testing are grouped into two sets, Class-I and Class-II, as described below.

Class-I Traces. We obtained four instances of malware from the internet: Bagle, IRCbot, Mybot and SDbot, and collected their traffic by infecting virtual machines hosts with each malware. The virtual hosts were all running the Windows XP Professional operating system with the same VMWare image file. Each run of traffic collection is one hour long, and includes the communications from eight instances of Bagle, three instances of IRCbot, five instances of Mybot, or five instances of SDbot. These numbers of instances were chosen to represent a very small fraction of the total campus hosts, specifically at most 0.0097% based upon the number of campus hosts observed sending traffic in the busiest hour, which has over 33,000 distinct IP addresses. The characteristics of these malware are described in Appendix A.

For testing, we overlaid flows from these malware instances onto one hour of our recorded campus network traffic, and assigned the malware traffic to originate from randomly selected internal hosts observed to be active during that hour. This makes our testing scenario much more realistic, since the internal hosts to be identified still exhibit their normal connection patterns, in addition to subtle malware activities.

Class-II Traces. We also obtained network traces of botnets gathered from honeynets, including an IRC-based Spybot, a HTTP-based botnet (similar to the Bobax worm[3]),

[2] Since Argus records are bi-directional, the source and destination IP addresses are swappable in the logic that matches packets to flows. However, the source IP address in the record is set to the IP address of the host that initiated the connection.

[3] `http://www.secureworks.com/research/threats/bobax/`

and a large IRC botnet captured in the wild. The Spybot trace contains communications from four bots for the duration of 32 minutes; the HTTP-bot trace contains communications from four bots over the course of three hours; and the large botnet trace contains traffic from more than three hundred bots over seven minutes.

These botnet traces were then overlaid onto each hour of our recorded campus traffic, in the same way as the Class-I traces. For the trace that spans multiple contiguous hours, i.e., the HTTP botnet trace, we overlaid it onto the same number of contiguous hours in the campus network traffic, performed analysis on each of the hours "covered" by the malware trace, and reported the hour that TĀMD detected the malware aggregate. This time window was then shifted by one hour, and the experiment repeated until we reached the end date of our campus traffic collection.

In our initial tests, we found that these malware-infected hosts were obscured by certain unknown hosts with highly unusual behavior, which turned out to be PlanetLab (`http://www.planet-lab.org`) and Tor (`http://tor.eff.org`) nodes. The experience of identifying these hosts and their exclusion from our dataset for the experiments reported in Section 5.2 is described in Appendix B. In practice, a system administrator can remove such hosts known for unusual behavior prior to performing analysis using TĀMD.

5.2 Detecting Malware

As described in Section 5.1, TĀMD was given all TCP and UDP traffic collected at the edge of our university network in hourly batches, overlaid with malware traffic assigned to randomly selected internal hosts. The same analysis steps were repeated for each hour over three weeks in November and December 2007.

The granularity of external destinations was set to be /24 subnets. While the communication records from the current hour were given to FindSuspiciousAggregates as Λ, the set Λ_{past} was selected from communication records in the past (specifically, from the beginning of our traffic collection dating to the first week of September 2007) that represented the general trend and the periodicity in the traffic. Specifically, Λ_{past} consisted of traffic from, in reference to the time frame for Λ, (i) the same hour from the same days of the week, (ii) the same hour from the same days of the month, (iii) the same hour from the previous two days, and (iv) the previous two hours. For example, if Λ consists of traffic from 2 to 3 PM on Wednesday, November 28th, then Λ_{past} will include traffic from 2 to 3 PM every Wednesday before that, from 2 to 3 PM in the previous two days (November 27th and 26th), and from 12 to 2 PM on November 28th.

In all experiments, TĀMD was able to identify all the infected hosts (with the exception of the Class-II large IRC trace, as described later) while the number of additional aggregates reported was only about 1.23 per hour on average. For the Class-II HTTP-botnet trace that spans multiple hours, TĀMD always detected the infected hosts in the very first hour. For the case of the Class-II large IRC botnet trace, which contains 340 infected bots, TĀMD was able to identify 87.5% of the bots on average, and these bots were all grouped in a single aggregate. We suspect that the reason not every bot in the botnet was detected is due to the randomness in our choice of selected internal hosts to which the malware traffic was assigned, such that a selected internal host that was

Malware traces	ByDest$^{\tau}$ (line 101)	ByPayload$^{\delta_{Ed}}$ (line 105)	ByPlatform (line 106)
Class-I			
Bagle	47.46 (± 23.13)	4.19 (± 2.34)	2.55 (± 1.33)
IRCbot	35.10 (± 20.51)	2.74 (± 1.41)	1.98 (± 0.98)
Mybot	45.60 (± 25.10)	3.19 (± 1.76)	2.13 (± 1.09)
SDbot	52.15 (± 43.87)	3.55 (± 1.88)	2.34 (± 1.16)
Class-II			
Spybot	39.18 (± 22.31)	2.95 (± 1.44)	2.04 (± 0.92)
HTTP bot	53.97 (± 26.54)	3.31 (± 1.91)	2.22 (± 1.21)
Large IRC bot	44.54 (± 16.16)	4.39 (± 2.75)	2.39 (± 1.32)

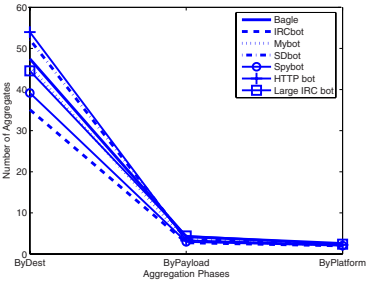

Fig. 2. Mean number of aggregates (± std. dev.) remaining after each function in Figure 1

also contacting other suspicious subnets (not relevant to the botnet) is likely to bias the dimension reduction and clustering algorithms.

Figure 2 shows for each malware experiment (the rows), the number of aggregates remaining after applying each aggregation function (the columns), averaged over all test hours. The number of aggregates is reduced after each aggregation function, as they become more refined to satisfy multiple characteristics. The single aggregate consisting solely of infected hosts was always identified, in every malware experiment. As shown in the figure, even for homogeneous networks where the majority of internal hosts are of the same platform, applying ByDest$^{\tau}$ and ByPayload$^{\delta_{Ed}}$ would still yield good results.

5.3 Unknown Aggregates

As indicated in Figure 2, our methodology detected a small number of unknown aggregates (about 1.23 per hour, on average) in addition to the one aggregate of infected hosts that we overlaid on the trace. We found that some of these same unknown aggregates regularly appeared for that hour of input data, across different malware experiments. Further investigation based on the 64 bytes of flow payload available to us, port numbers, and protocol field (for privacy reasons, the IP addresses were anonymized), showed that these aggregates included NetBIOS messages on port 137, DNS name server queries, SMTP connection timeout messages, and advertising-related HTTP requests; several of these suggest that additional investigation may be warranted. Others included connections to online game servers and large flows over high-order ports, which we suspect to be peer-to-peer (P2P) transfers. All of these aggregates consisted of internal hosts contacting rare sites, and often consisted of less than five hosts sharing one or two common destination subnets.

In theory, a group of internal hosts visiting a new popular website (i.e., the "slashdot" effect) could also form an aggregate. However, it is unlikely that all of the hosts would come from the same platform, and in our experiments, we believe we saw very few such aggregates. We thus believe that TĀMD is a useful data reduction tool for malware identification.

6 Discussion and Ongoing Work

Approaches by which malware writers might attempt to avoid detection by our techniques include encrypting their malware traffic, so that our payload comparisons will be ineffective. To accommodate encryption, our techniques can be generalized to define encrypted content (which itself is generally easy to detect) as "similar"; we are exploring the impact of this adaptation in ongoing work. Malware writers could go further and have their malware communicate steganographically, though at the cost of greater sophistication and lower bandwidth. Detecting steganographic communication is itself an active area of research (e.g., [31]) from which TĀMD could benefit.

A second way that malware writers could try to avoid detection by TĀMD is with alternative botnet architectures. Although the vast majority of spyware and botnets found today use a centralized IRC command-and-control server, other botnet architectures have been reported, such as P2P botnets (Phatbot[4], Trojan.Peacomm bot [12], Sinit P2P trojan[5]) or HTTP-based botnets (Clickbot.A [9]). Still others have been proposed, such as hybrid P2P and centralized botnets [41, 43].

Even among these alternative architectures, a large number exhibit characteristics that we believe should be detectable via FindSuspiciousAggregates in Section 4. For example, Trojan.Peacomm bots, while using a P2P network to transfer addresses of compromised web servers among them, still connect to these web servers to download malicious executables for sending spam or performing DoS attacks. This activity of collectively contacting web servers matches the behavior that our techniques successfully detected in our evaluations. The same detection method can also be applied to HTTP-based bots, such as Clickbot.A [9], which commit click frauds by having bots connect to a compromised web server for a list of websites and search keywords, or for a URL to download updated bot versions. Vogt et al. [41] suggested a "super-botnet", where the botnet is composed of individual smaller centralized botnets, and the controllers from each smaller botnet peer together in a P2P network. Since the individual smaller botnets still use a centralized architecture, this should be still be detectable via our techniques. Wang et al. [43] proposed a hybrid P2P botnet where each bot maintains its own peer list and polls other bots periodically for new commands. However, in order to monitor the IP address and resources of each individual bot, the botnet supports a command by which the botmaster can solicit all bots to report to a specific compromised server. Again, this behavior should be detectable by FindSuspiciousAggregates.

That said, some P2P bots avoid contacting a common server for the transfer of executables or other tasks, such as Phatbot and the Sinit trojan. While Phatbots find peers by registering themselves as Gnutella clients, the Sinit trojan sends out random probes for peer discovery. In both cases, forming aggregates based on payload similarity should remain effective, provided that similarity is generalized as described above to accommodate encrypted traffic (which Phatbot utilizes). Similarly, platform-based aggregation should also be effective, as both are platform-dependent. We are evaluating these directions in ongoing work, as well as alternative aggregation methods to help identify these types of malware.

[4] See `http://www.secureworks.com/research/threats/phatbot`
[5] See `http://www.secureworks.com/research/threats/sinit`

7 Conclusion

In this paper, we presented TĀMD, a system that identifies hosts within a network that are possibly infected by stealthy malware by finding those that share common and unusual network communications. TĀMD employs three aggregation functions to group hosts based on the following characteristics. First, the destination aggregation function, ByDest^{τ}, forms aggregates of internal hosts that contact the same combination of busier-than-usual external destinations. A binary vector is formed for each internal host, with each dimension representing one of the selected external destinations. The vectors are processed by PCA for dimension reduction, and clustered by K-means clustering. New clusters are selected as those that do not conform to preceding communication patterns. Second, the payload aggregation function, $\mathsf{ByPayload}^{\delta_{\mathsf{Ed}}}$, identifies communications with similar payloads in terms of a type of edit distance. This is done by first embedding the payload strings into vectors in L1 space, and then finding close vectors by an approximate nearest-neighbor algorithm. Third, the platform aggregation function, ByPlatform, forms aggregates that involve hosts running on common platforms, as inferred using TTL values or platform-specific sites to which they connect.

We detailed a configuration of TĀMD that employs these functions in combination to identify platform-dependent malware infections that report to common sites. A common site might be an IRC channel for receiving commands, a public webserver for downloading binaries, a denial-of-service victim they are instructed to attack, or a database server for uploading stolen information, as is typical of most bots and spyware. Our experiments showed that, with traffic generated from real spyware and bot instances, this configuration of TĀMD reliably extracted malware traffic from all traffic passing the edge of a university network, while the number of other aggregates reported is very low. This is achieved even in tests where the number of simulated infected hosts comprised only about 0.0097% of over 33,000 internal hosts in the network.

Acknowledgements

We are grateful to Moheeb Rajab and other members of the Johns Hopkins Honeynet Project (`http://hinrg.cs.jhu.edu/jhuhoneynet/`) for providing malware binaries that we used in our evaluations, and to Wenke Lee, Guofei Gu, David Dagon, and Yan Chen for providing botnet traces. We are also grateful to Chas DiFatta, Mark Poepping and other members of the EDDY Initiative (`http://www.cmu.edu/eddy/`) for facilitating access to the network traffic records from Carnegie Mellon University used in this research. This research was supported in part by NSF awards 0326472 and 0433540.

References

[1] Aiello, W., Kalmanek, C., McDaniel, P., Sen, S., Spatscheck, O., Van der Merwe, J.: Analysis of communities of interest in data networks. In: Proceedings of Passive and Active Measurement Workshop (2005)

[2] Bächer, P., Holz, T., Kötter, M., Wicherski, G.: Know your enemy: Tracking botnets. Technical report, The Honeynet Project and Research Alliance (2005)
[3] Barford, P., Kline, J., Plonka, D., Ron, A.: A signal analysis of network traffic anomalies. In: Proceedings of ACM SIGCOMM Internet Measurement Workshop (2002)
[4] Binkley, J.R., Singh, S.: An algorithm for anomaly-based botnet detection. In: Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (2006)
[5] Brownlee, N., Mills, C., Ruth, G.: Traffic flow measurement: Architecture. RFC 2722 (1999)
[6] Cheng, D., Kannan, R., Vempala, S., Wang, G.: A divide-and-merge methodology for clustering. ACM Transactions on Database Systems 31(4) (2006)
[7] Cooke, E., Jahanian, F., McPherson, D.: The zombie roundup: Understanding, detecting, and disrupting botnets. In: Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (2005)
[8] Cormode, G., Muthukrishnan, S.M.: The string edit distance matching problem with moves. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (2002)
[9] Daswani, N., Stoppelman, M.: The Google Click Quality, and Security Teams. The anatomy of clickbot.A. In: Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (2007)
[10] Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Symposium on Computational Geometry (2004)
[11] Goebel, J., Holz, T.: Rishi: Identify bot contaminated hosts by IRC nickname evaluation. In: Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (2007)
[12] Grizzard, J.B., Sharma, V., Nunnery, C., Kang, B.B., Dagon, D.: Peer-to-peer botnets: Overview and case study. In: Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (2007)
[13] Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the USENIX Security Symposium (August 2008)
[14] Gu, G., Porras, P., Yegneswaran, V., Fong, M., Lee, W.: BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In: Proceedings of the USENIX Security Symposium (2007)
[15] Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic. In: Proceedings of the 2008 ISOC Network and Distributed System Security Symposium (February 2008)
[16] Handelman, S., Stibler, S., Brownlee, N., Ruth, G.: New attributes for traffic flow measurement. RFC 2724 (1999)
[17] Jolliffe, I.T.: Principal Component Analysis. Springer, Heidelberg (1986)
[18] Karamcheti, V., Geiger, D., Kedem, Z., Muthukrishnan, S.M.: Detecting malicious network traffic using inverse distributions of packet contents. In: Proceedings of the ACM SIGCOMM Workshop on Mining Network Data (2005)
[19] Karasaridis, A., Rexroad, B., Hoeflin, D.: Wide-scale botnet detection and characterization. In: Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (2007)
[20] Kaufman, L., Rousseeuw, P.J.: Finding Groups in Data. An Introduction to Cluster Analysis. Wiley, Chichester (1990)
[21] Kim, H., Karp, B.: Autograph: Toward automated, distributed worm signature detection. In: Proceedings of the USENIX Security Symposium (2004)
[22] Kim, S.S., Reddy, A.L.N., Vannucci, M.: Detecting traffic anomalies using discrete wavelet transform. In: Proceedings of the International Conference on Information Networking (2004)

[23] Kohler, E., Li, J., Paxson, V., Shenker, S.: Observed structure of addresses in IP traffic. IEEE/ACM Transactions on Networking 14(6) (2006)
[24] Kreibich, C., Crowcroft, J.: Honeycomb - creating intrusion detection signatures using honeypots. In: Proceedings of the ACM SIGCOMM Workshop on Hop Topics in Networks (2003)
[25] Lakhina, A., Papagiannaki, K., Crovella, M.: Structural analysis of network traffic flows. In: Proceedings of ACM SIGMETRICS/Performance (2004)
[26] Livadas, C., Walsh, B., Lapsley, D., Strayer, T.: Using machine learning techniques to identify botnet traffic. In: Proceedings of the IEEE LCN Workshop on Network Security (2006)
[27] Moore, D., Voelker, G.M., Savage, S.: Inferring internet denial-of-service activity. In: Proceedings of the USENIX Security Symposium (2001)
[28] Newsome, J., Karp, B., Song, D.: Polygraph: Automatic signature generation for polymorphic worms. In: IEEE Security and Privacy Symposium (2005)
[29] Parekh, J.J., Wang, K., Stolfo, S.J.: Privacy-preserving payload-based correlation for accurate malicious traffic detection. In: Proceedings of the ACM SIGCOMM Workshop on Large Scale Attack Defense (2006)
[30] Perdisci, R., Gu, G., Lee, W.: Using an ensemble of one-class SVM classifiers to harden payload-based anomaly detection systems. In: Proceedings of the International Conference on Data Mining (2006)
[31] Provos, N., Honeyman, P.: Detecting steganographic content on the Internet. In: Proceedings of the 2002 ISOC Network and Distributed System Security Symposium (NDSS) (February 2002)
[32] Racine, S.: Analysis of Internet Relay Chat Usage by DDoS Zombies. Master's thesis, Swiss Federal Institute of Technology Zurich (2004)
[33] Rajab, M.A., Zarfoss, J., Monrose, F., Terzis, A.: A multifaceted approach to understanding the botnet phenomenon. In: ACM SIGCOMM/USENIX Internet Measurement Conference (2006)
[34] Ramachandra, A., Feamster, N., Vempala, S.: Filtering spam with behavioral blacklisting. In: Proceedings of the ACM conference on Computer and Communications Security (2007)
[35] Ramachandran, A., Feamster, N.: Understanding the network-level behavior of spammers. In: Proceedings of ACM SIGCOMM (2006)
[36] Ramachandran, A., Feamster, N., Dagon, D.: Revealing botnet membership using DNSBL counter-intelligence. In: Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet (2006)
[37] Reiter, M., Yen, T.: Traffic aggregation for malware detection. Technical Report CMU-CyLab-07-017, Carnegie Mellon University (2007)
[38] Singh, S., Estan, C., Varghese, G., Savage, S.: Automated worm fingerprinting. In: Proceedings of the Symposium on Operating Systems Design and Implementation (2004)
[39] Taylor, C., Alves-Foss, J.: NATE - network analysis of anomalous traffic events, a low-cost approach. In: Proceedings of the New Security Paradigms Workshop (2001)
[40] Terrell, J., Zhang, L., Zhu, Z., Jeffay, K., Shen, H., Nobel, A., Donelson Smith, F.: Multivariate SVD analyses for network anomaly detection. In: Poster Proceedings of ACM SIGCOMM (2005)
[41] Vogt, R., Aycock, J., Jacobson Jr., M.J.: Army of botnets. In: Proceedings of the Network and Distributed System Security Symposium (2007)
[42] Wang, K., Parekh, J.J., Stolfo, S.J.: Anagram: A content anomaly detector resistant to mimicry attack. In: Proceedings of the 9th International Symposium on Recent Advances in Intrusion Detection (2006)
[43] Wang, P., Sparks, S., Zou, C.C.: An advanced hybrid peer-to-peer botnet. In: Proceedings of the 1st Workshop on Hot Topics in Understanding Botnets (2007)

A Class-I Malware Instances

For our testing described in Section 5, traffic from four malware instances was collected using virtual machine hosts infected with each malware. The virtual hosts were all running the Windows XP Professional operating system with the same VMWare image file. Each run of traffic collection is one hour long.

Bagle[6] is spyware that, on execution, runs as a background process and attempts to download other malicious executables from various sites, while generating pop-up windows and hijacking the web browser to advertising websites. As with other types of spyware and adware, Bagle initiates connections to numerous destinations that are set up to exclusively host advertisements or other malicious content. We collected Bagle traffic by simultaneously running eight instances of Windows XP virtual machine hosts infected with Bagle.

IRCbot[7] is a backdoor trojan that connects to an IRC server and waits for commands from the attacker. In addition, after successfully connecting to the command-and-control center, the bot downloads an update executable from a designated webserver, and goes on to scan the local /16 subnet attacking other machines with the LSASS vulnerability on port 445[8] and the NetBIOS vulnerability on port 139[9]. We collected traffic from two instances of IRCbot running on two Windows XP virtual machine hosts.

Mybot[10] is spyware, a worm, and a bot that connects to an IRC server to wait for commands, and also records keystrokes and steals other personal information on the victim host. This malware is especially subtle in its communications. When it is only waiting for commands on the IRC server, the bot initiates one connection every 90 seconds, in the form of IRC PING/PONG messages. In the hour of our traffic collection, Mybot simply waited for commands on the IRC channel, and its only outbound connections were these PING/PONG messages. We collected traffic for five Mybot instances.

SDbot[11] is a trojan and a bot that opens a back door to connect to an IRC server. Similar to Mybot, when it is waiting for commands from the attacker, SDbot only makes outbound connections once every 90 seconds, in the form of IRC PING/PONG messages. We collected SDbot traffic from simultaneously running five instances of Windows XP virtual machine hosts infected with this malware.

B Outlier Hosts

In the early stages of our analysis described in Section 5, we found that often TĀMD failed to detect the malware-laden hosts, but rather identified other internal hosts as

[6] http://www.trendmicro.com/vinfo/virusencyclo

[7] http://www.symantec.com/enterprise/security_response/threatexplorer/threats.jsp

[8] http://www.microsoft.com/technet/security/Bulletin/MS04-044.mspx

[9] http://msdn2.microsoft.com/en-us/library/ms913275.aspx

[10] http://www.sophos.com/security/analyses/w32rbotxf.html

[11] http://www.symantec.com/enterprise/security_response/threatexplorer/threats.jsp

more symptomatic of malware. Upon further inspection, we identified the internal hosts that resulted in these false alarms: PlanetLab nodes (`http://www.planet-lab.org`) and a Tor node (`http://tor.eff.org`).

In the case of PlanetLab nodes, we noticed that during the destination aggregation function, the vectors after PCA analysis often had very low dimensionality, e.g., two, where two principal components were able to cover over 90% of the data variance. Clustering these vectors resulted in a few outliers forming their own individual clusters, unlike any of the other vectors in Λ (i.e., the "new vectors"), or even those from Λ_{past} (the "old vectors"). This is shown in Figure 3. The two axes correspond to the top two principal components on which the original data is projected. The outliers were found to be PlanetLab nodes, which, being a development and testing platform, exhibit behavior deviating from other hosts. Their existence was also the reason why PCA analysis was able to reduce the vector dimensionality down to only two, since PlanetLab nodes' behavior is so different from other hosts that only two principal components were needed to capture most of the data variance.

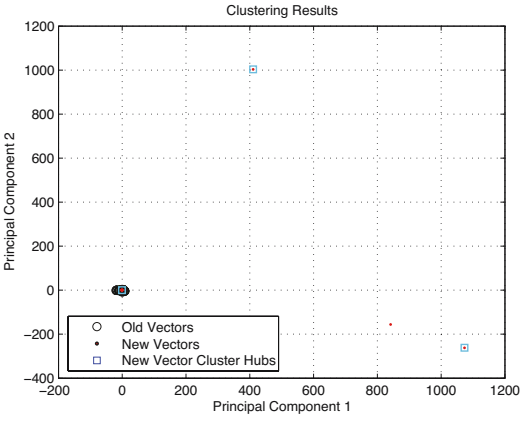

Fig. 3. Clustering results after dimension reduction by PCA. The three outliers were found to be PlanetLab nodes.

In another example from experiments involving the Bagle trojan spyware, we noticed that even though TĀMD was able to form a final aggregate containing all spyware traffic and spyware traffic only, at times it also combined another unknown host into the spyware-hosts aggregate, both in the ByDest and the ByPayload functions. Similar investigations revealed that this additional node is a Tor router inside the campus network. Tor offers online anonymity by routing packets over random routes between Tor servers so that the source and destination of the packet is obfuscated. Because the traffic comes from different anonymous hosts, it is possible that, even though the Tor router itself is not infected, another host routing traffic through the Tor node may be a spyware victim.

For this work, we removed PlanetLab and Tor nodes from our analysis.

C Performance

The top half of Table 2 shows the run times in seconds for each aggregation function and for each malware instance, averaged over the week's worth of traffic (in one-hour intervals) we used to performed our experiments. In our present implementation of TĀMD, ByDest$^{\tau}$ is implemented in Matlab, and ByPayload$^{\delta_{\mathsf{Ed}}}$ and ByPlatform are

Table 2. Mean run times of each phase in seconds of algorithm in Figure 1 and means of measures impacting performance (± std. dev.)

Malware traces	ByDest$^{\tau}$ (line 101)	ByPayload$^{\delta_{\mathsf{Ed}}}$ and ByPlatform (lines 105, 106)	Total time	Size of SuspiciousSubnets	Internal hosts contacting SuspiciousSubnets
Class-I					
Bagle	79.48 (± 264.54)	14.08 (± 18.07)	93.48 (± 271.51)	701.87 (± 596.78)	754.73 (± 812.75)
IRCBot	94.67 (± 350.13)	20.19 (± 16.78)	114.86 (± 356.72)	927.23 (± 561.33)	742.13 (± 836.55)
Mybot	63.82 (± 177.34)	10.96 (± 15.28)	70.93 (± 183.43)	686.03 (± 565.11)	728.45 (± 708.65)
SDbot	102.34 (± 355.25)	10.21 (± 19.51)	112.55 (± 359.23)	749.01 (± 577.49)	952.96 (± 1191.66)
Class-II					
Spybot	86.30 (± 276.81)	63.42 (± 38.56)	151.15 (± 286.99)	850.14 (± 609.19)	777.71 (± 848.43)
HTTP bot	83.12 (± 278.76)	15.75 (± 20.62)	99.31 (± 287.11)	697.36 (± 609.15)	776.76 (± 848.43)
Large IRC Bot	110.64 (± 253.78)	46.00 (± 34.78)	156.64 (± 260.42)	760.83 (± 548.48)	1104.42 (± 799.58)

implemented in C. For the numbers reported in Table 2, ByDest$^{\tau}$ was run on a PC with a Pentium IV 3.2 GHz processor and 3 GB of RAM, and ByPayload$^{\delta_{\mathsf{Ed}}}$ and ByPlatform were run on a Dell PowerEdge server with dual core 3 GHz processors and 4 GB of RAM.

The running times of the aggregation functions depend on several factors, including the number of external destinations identified as suspicious (i.e., SuspiciousSubnets as computed by ByDest$^{\tau}$) and the number of flows to those suspicious destinations; averages for these numbers are also listed in Table 2. The amount of traffic in Λ_{past} is especially critical to the performance of ByDest$^{\tau}(\Lambda, \Lambda_{\mathsf{past}})$, since it accesses significant amounts of historical data (i.e., Λ_{past}) to define the "normal" behavior for this network. While the implementation of TĀMD is not yet optimized, retrieving historical data from the database contributed to the majority of the slowdown. This problem can be alleviated in the future by performing these calculations in advance and storing them statically, only updating incrementally as more data is collected.