Early and Robust Malware Detection in Enterprise Networks

Mkhail Kazdagli
University of Texas at Austin
mikhail.kazdagli@utexas.edu

Constantine Caramanis
University of Texas at Austin
constantine@utexas.edu

Sanjay Shakkottai
University of Texas at Austin
shakkott@austin.utexas.edu

Mohit Tiwari
University of Texas at Austin
tiwari@austin.utexas.edu

***Abstract*—Behavioral malware detectors, by using statistical methods, promise to expose previously unknown malware and are an important security primitive. However, even the best detectors suffer from high false positives and negatives. In this paper, we address the challenge of composing a community of weak per-device behavioral detectors (local detectors or LDs) in *noisy* communities (i.e. that produce alerts at unpredictable rates) into an accurate and robust global anomaly detector (GD).**

Our system – Shape GD – combines two insights: *Structural:* actions such as visiting a website (waterhole attack) or membership in a shared email thread (phishing attack) by nodes correlate well with malware spread, and create dynamic *neighborhoods* of nodes that were exposed to the same attack vector; and *Statistical:* feature vectors corresponding to true and false positives of local detectors have markedly different conditional distributions – i.e. their histogram *shapes* differ.

Unlike prior works that aggregate local detectors' alert bitstreams, Shape GD analyzes the *feature vectors that led to local alerts*. Shape GD first *filters* these alert feature vectors along neighborhood lines, efficiently maps a neighborhood's FVs' statistical shapes into a scalar score ('ShapeScore'), and trains on benign program traces (e.g., from developers' test inputs) to learn the ShapeScore of false positive neighborhoods.

We evaluate Shape GD through emulation on a community of Windows systems, using system call traces from a few thousand malware and benign applications and simulating a phishing attack in a corporate email network and a waterhole attack through a popular website. In both these scenarios, we show that Shape GD detects malware early ($\sim$10–100 nodes in $\sim$1000–100K size neighborhoods respectively) and robustly ($\sim$100% global TP and $\sim$1% global FP rates).

I. Introduction

Behavioral detectors are a crucial line of defense against malware. By extracting features out of network [1]–[4], system calls [5]–[7], instruction set [8], [9], and hardware [10]–[12] level actions, behavioral detectors train machine learning algorithms to classify program binaries [13] and executions [10], [11] as either malicious or benign. Behavioral detectors are a widely deployed defensive technique [14] in fast-changing environments where new systems and sensors drive previously unseen 'zero-day' malware that bypass known threat models and formal specifications – such as abuse of new permissions in Android, row-hammer attacks on DRAMs, accelerometers in addition to bugs in trusted codebases.

However, behavioral detectors are *weak* – i.e., have high false positives and negatives. One reason is that a large class of benignware and malware have very similar behaviors – such as accessing users' files, encrypting users' data and code at run-time, and making web/HTTP requests. Another reason for weak detectors is that malware developers adapt to deployed detectors – e.g., by generating malware that specifically evades behavioral detectors [15], [16]. Deploying behavioral detectors can thus create a large stream of alerts that drive expensive program analyses or human analysts – boosting weak malware detectors is thus an important problem.

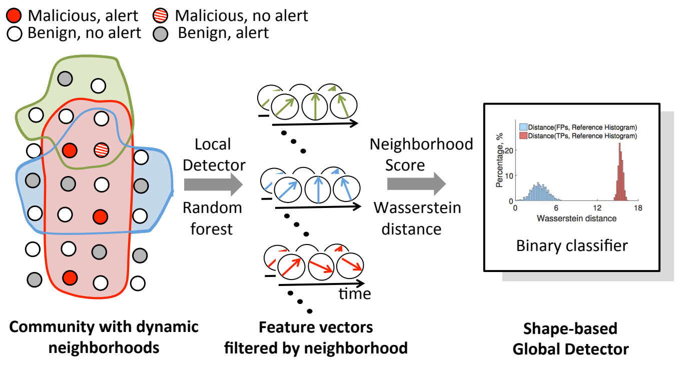

Fig. 1. (L to R) Each circle is a node that runs a local malware detector (LD). Our goal is to create a robust global detector (GD) from weak LDs. We observe that nodes naturally form *neighborhoods* based on attributes relevant to attack vectors – e.g., all client devices that visit a website W within the last hour belong to neighborhood NB_w, or all users who received an email from a mailing list M in the last hour belong to neighborhood NB_m. We propose (a) a GD that analyzes *feature vectors* (FVs) from each node, instead of only LD alerts, and further *filters* the FVs based on neighborhoods; and (b) a 'Shape GD' algorithm that exploits a new insight – the conditional distribution of true positive FVs differs from false positive FVs – to robustly classify neighborhoods as malicious.

Much prior research has focused on improving malware detectors deployed *locally* on each machine (e.g., [5], [17], [18]). This includes engineering better features (n-grams, histograms, markov models etc of system calls and network traffic) and composing them using ensemble methods [12] – our results show that even the best local detectors (LDs) based on prior work [5] have $\sim$ 6% false-positives and 92.4% true-positives.

Complementary research proposes a *global* detector (GD) that uses the outputs of a *community* of local detectors to boost the detection and false positive rate [19]. Such *collaborative* intrusion detection systems (CIDSs) rely on global detectors to generate a global alert if a significant number of local detectors are raising an alert – several variants of this *count*-based idea (e.g. count number of alerts, search for sizeable clusters in feature space) [20], [21] have been studied and are deployed by security companies such as Cisco [22], Dell [23], etc.

Challenge – Early and Robust CIDS. In early stages of infection, alerts generated by a community of LDs are masked by the weak detectors' false positives. Fundamentally, this is because the noise floor (and its variance) is large due to the LDs' high false positive rates. Further, count-based GDs become dramatically inaccurate when the underlying community is even slightly *noisy* – i.e., communities where the total number of feature vectors within a time window varies unpredictably. For example, a GD that monitors visits to risky or uncommon websites will miss feature vectors from employee-devices that visit the risky websites from outside the corporate network (i.e., GD makes an over-estimation error if calibrated with the employee-device in the network, or an under-estimation error if calibrated without the employee-device). Further, employees that opt to not report detailed logs due to privacy concerns, devices that go out of range, hardware or network failures, etc can all make GD's estimates of community-level feature vectors noisy.

Any noise in estimating the number of feature vectors in the community *linearly* affects global decision thresholds and hence has a significant affect on the global false positive/negative rate. This linear scaling is debilitating – for example, our case studies of phishing and waterhole attacks show that underestimating community size by even 2% (or overestimating by 14%) leads to almost 100% false positives (respectively, almost 0% true positives) in a count-based GD system (Section VI-E). Our goal thus is to aggregate *weak* LDs in *noisy* communities in a robust manner.

Proposed Ideas – Neighborhood filtering and Shape GD. CIDSs aggregate a *community* of local detectors. We define a community as a set of nodes that are loosely correlated based on real-world attributes such as a common employer or occupation (e.g., all employees in a company), membership in a mailing list or work group (such as a department in an enterprise) or even a social network group. In our setting, the CIDS thus does not need to know community sizes precisely and one of many community detection algorithms are admissible [24], [25].

Within a community, we introduce a finer-grained notion of a *neighborhood* – a set of nodes that share an *action attribute* such as having visited a common website or received emails from the same source within a *neighborhood time window* (NTW). *Action attributes* that determine neighborhoods are defined statically by an analyst based on common *attack vectors* – neighborhoods are then instantiated dynamically at run-time. Thus, neighborhoods are dynamic sub-communities of nodes that are likely to be exposed to a similar attack vector – e.g., a compromised web-server or a malicious phishing email (see Figure 1 for an illustration with three neighborhoods).

Neighborhood filtering. For *early* detection of malware spread, we propose that the GD aggregate LDs' outputs per-neighborhood instead of per-community. An attack vector – such as a popular web-server used to distribute exploits in a waterhole attack – is more likely to exploit nodes along neighborhood lines – i.e., nodes that visited the compromised server in the current time window – compared to an arbitrary node in the community that may get compromised in later stages of an infection. A similar argument can be made for phishing attacks – a neighborhood of nodes that received emails from a common source in the current NTW are more likely to be compromised (true positives) than arbitrary nodes (that are more likely to be false positives). Neighborhood filtering exploits this latent structure that creates dynamic neighborhoods within a community. Most importantly, since neighborhoods are smaller than the overall community, we show that a GD can identify infected neighborhoods as anomalies much quicker than identifying the entire community as anomalous. At the same time, neighborhoods are *even more noisier* to estimate than communities — this motivates our Shape GD algorithm.

Shape GD. We propose a new GD algorithm ('Shape GD') that analyzes feature vectors (FVs) that lead to local detector alerts – *alert-FVs* – instead of operating only on the LDs' time-series of 1-bit alerts. Our key insight is that a GD can separate true positive neighborhoods from false positive ones by comparing the *distributional shape of alert-FVs* from each neighborhood. Specifically, Shape GD does *not* look at all FVs generated in a neighborhood, but only those that cause alerts by the LDs. *This* **alert filtering***, we show, has two key properties: (i) the distribution of the alert-FVs strongly separates malicious and benign neighborhoods (essentially, it separates the true positive alert-FVs from false positive alert-FVs), and (ii) is robust to noise in the neighborhood size estimates.*

Such a *shape-based GD* requires a quantitative score function that maps a set of alert-FVs from a neighborhood into a scalar value (the neighborhood's 'ShapeScore') that can then be used to train a GD classifier. We propose an efficient method to compute ShapeScores, and show that (given sufficient FV samples) our Shape GD can detect malicious neighborhoods with less than 1.1% and 2% compromised nodes per neighborhood (in two case studies involving waterhole and phishing attacks respectively), at a false positive and true positive rate of 1% and 100% respectively.

Contributions. To summarize, neighborhood filtering enables structural information about attack vectors to be captured in an algorithmically amenable setting – while Shape GD separates conditional FV distributions from variable-sized neighborhoods to identify the ones that show early stages of malware infection. Our specific contributions are as follows.

- Neighborhood filtering and Shape GD algorithms that exploit a new property – the statistical 'shape' of a neighborhood separates the ones with true positives from those with false positives – for early and robust malware detection in noisy neighborhoods.
- An efficient CIDS – comprising random forest LDs and a Shape GD that computes ShapeScores using the Wasserstein distance between neighborhoods' alert-FV distributions and a reference distribution (built on false positive FVs) – that can identify malicious

neighborhoods using only 15,000 FVs (roughly 15 seconds of FV data from a 1000-node neighborhood).

- Phishing case study. Shape GD detects a phishing attack with 1% false positive rate in a medium size enterprise network with a neighborhood of 1086 nodes when only 17.08 nodes (using temporal neighborhoods) and 4.48 nodes (with additional mailing-list based structural filtering) are infected.
- Waterhole attack case study. Shape GD detects a waterhole attack with 1% false positive rate when only 107.5 nodes (using temporal neighborhoods) and 139.9 (with additional server specific structural filtering) out of possible $\sim 550,000$ nodes are infected.

We finally remark that the LD and GD false positives (FPs) have very different interpretations. In a phishing attack, an LD FP of 1% in a neighborhood of 1000 nodes means that we will get about 10 FP alerts per second. The Shape GD, on the other-hand, uses these LD FP alerts for decision making. Thus a GD false positive occurs when it misclassifies a collection of LD alerts – a much rarer scenario.

Specifically, a GD FP rate of 1% means that in our phishing attack scenario, we will receive a global false alert about once every 100 - 300 hours. Similarly, in the waterhole scenario a global false positive occurs every 100 sec. Comparing the number of LDs' FPs that are reported to a GD in a Count GD v. Shape GD, temporal neighborhood filtering reduces total FPs by ~100× (phishing) and ~200× (waterhole), while adding structural filtering reduces total FPs by ~1000× and ~830× respectively (see Section IX for details).

II. Overview of Shape GD

We begin with a baseline CIDS that deploys local detectors (LDs) at each device and a global detector (GD) that receives alerts and other metadata from the local detectors. The LD at each node transforms its input signal into an alert time series. This transformation consists of two steps: *(a) Generate Feature Vectors:* convert the raw OS system calls trace into a feature vector (FV) time series, and *(b) Generate Alerts:* Determine if each FV is malicious or not using a local detector (typically through random forests, SVM, etc.).

Inferring neighborhoods from common attack vectors. Shape GD operates over dynamic neighborhoods (updated once every NTW). Neighborhoods are a set of nodes that share a statically defined *action attribute* – this allows an analyst to create neighborhoods of nodes based on common attack vectors. Below are three illustrative examples of communities and neighborhoods.

1. Waterhole attack. The community here consists of the employees of an enterprise such as Anthem Health [26], [27]. In a waterhole attack, adversaries compromise a website commonly visited by such employees as a way to infiltrate the enterprise network and then spread within the network to a privileged machine or user. Within this community, the neighbor of a node (user/machine) is simply all other nodes that visited the same websites within the current neighborhood time window (NTW). Specifically, if nodes A and B visit website X within the current neighborhood time window, they are said to be neighbors. In other words, all nodes that visit website X within the current NTW constitute a neighborhood (NB_x).

2. Phishing attack over enterprise email networks. The community here consists of all employees within an enterprise (e.g. system administrators managing a healthcare network). A phishing attack here would typically spread over email and use a malicious URL to lure nodes (users) to drive-by-download attacks [28], [29]. Here, a specific user's neighbors are that subset of users with whom she/he exchanged emails with during the current neighborhood time window (NTW).

3. Physical hardware attack. A community here consists of all machines in a workplace that are physically proximal (e.g. machines in a specific bank-branch). The potential attack mode here is through physical hardware such as badUSB. The neighborhood of a node is simply all other nodes that were connected to similar external hardware (e.g. a USB drive) over the current neighborhood time window.

Similar correlations (leading to natural notions of community and neighborhood) exist in attacks that target specific app-stores (e.g., the key-raider attack in the Cydia app-store or the malicious Xcode attacks due to compromised mirror sites) – these attacks also affected users with specific attributes (membership in a store or downloaded Xcode from specific sites) more likely than a random user in the network.

These popular attack scenarios, while different in their infection mechanisms, all exhibit community and neighborhood correlations – *if a user has malware, his/her neighbor is more likely to have malware than a randomly chosen node.*

A. Intuition behind Shape GD

There are two parts to our algorithm: *(i)* the neighborhood correlations matter – common actions and attack vectors statistically correlate nodes and thus implicitly define suspicious neighborhoods, and *(ii)* given these neighborhoods, the statistical shape of local detector false positives (FP conditional distribution) differs from the corresponding shape for true positives (TP conditional distribution). To set the stage for our algorithm, we consider a stylized statistical inference example. Suppose that we have an unknown number of nodes within a neighborhood. We want to distinguish between two extremes – all nodes only run benign applications (benign hypothesis), or all nodes are running malware (malware hypothesis). We look at a single snapshot of time where each node generates exactly one feature vector. Under the benign hypothesis, assume that the feature vector from each node is a (scalar valued) sample from a standard Gaussian with mean of '-1'; alternatively it is standard Gaussian with mean of '+1' under the malware hypothesis.

(a) Noisy local detectors: Given one sample (i.e., FV from one node), the best "local detector" is a threshold test: is the sample's value above zero or below? For this example, the probability of a false positive is (about) 15%.

(b) Aggregating local detectors over neighborhoods: Suppose there are 100 nodes and all of them report their value, and we are told that 90 of them are greater than 0 (i.e., 90 of the local detectors generate alerts). In this case, the expected number of alerts under the benign hypothesis is 15; and this would be 85 under the malware hypothesis. Thus, we can

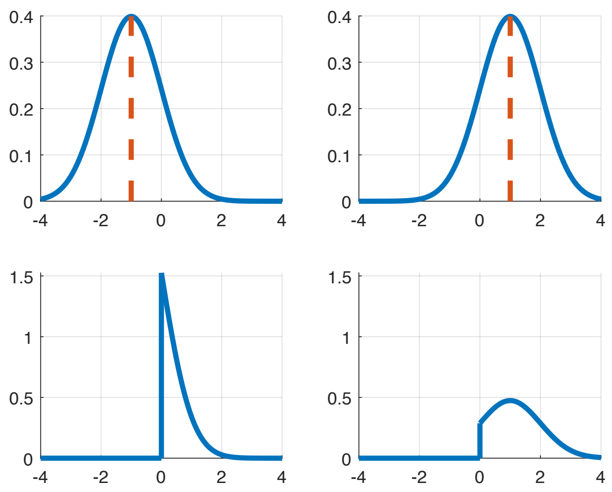

Fig. 2. (Shape of conditional distributions) The top left figure is the probability density function (pdf) of a benign FV, here a Gaussian with mean '-1'; and the top right figure is the pdf for a malware FV, here a Gaussian with mean'+1'. The optimal local detector at any node would declare 'malware' if a sample's value is positive, and declare 'benign' if a sample's value is negative. The bottom plots shows the pdfs of the same Gaussians but now conditioned on the event that the sample is positive. This is the pdf of the filtered FV, i.e, a sample that is chosen to be positive (and negative samples are deleted). This filtering corresponds to the action of the local detector at each node, which tags a sample as 'malware' if and only if the sample is positive. After LD filtering, note that the pdfs (corresponding to FP and TP FVs respectively) have different shapes.

conclude with overwhelming certainty (10^{-75} chance of error) that these samples are drawn from a malware neighborhood. This corresponds to a conventional threshold algorithm that count the number of alarms in a neighborhood and compares with a global threshold (here this threshold is 50).

(c) Count without knowing neighborhood size: Suppose, now, that we do not know the number of nodes (i.e., neighborhood size is unknown), and only know that there are a total of 90 alerts. In other words, out of the neighborhood of nodes, some 90 of them whose samples were positive reported so. What can we say? Unfortunately we cannot say much – if there were 100 nodes in neighborhood, then malware is extremely likely; however, if there were 1000 total nodes, then with 90 alerts, it is by far (exponentially) more likely that we have no infection. Because we do not know the neighborhood size, the global threshold cannot be computed.

(d) Robustness of Shape: While the number of alerts alone is uninformative, we can resolve whether the neighborhood is a 'false positive' or 'true positive' by considering the actual values of the 90 random variables corresponding to these *alerts*. These values represent independent draws from a *conditional* distribution – either the distribution of a normal random variable of mean '-1' *conditioned on taking a nonnegative value*, or the distribution of a normal random variable of mean '$+1$' *conditioned on taking a nonnegative value* (see Figure 2). This conditioning occurs because of the local detector – recall it tags a sample as an alert if and only if the sample drawn was nonnegative (optimal LD in this example). Thus, irrespective of the size of the neighborhood, the global detector would "look at the shape" of the empirical distribution (i.e. the distribution constructed from the received samples) of the received samples (FVs). If this were "closer" to the left rather than the right plot in Figure 2, it would declare "uninfected"; other-wise declare "infected".

B. *From Intuition to Algorithm Design*

While the simple example highlights the resolving power of conditional distributions of feature vectors for distinguishing between TPs and FPs, to use this insight in practical CIDSs, we need to address two issues: *(i)* While the two figures in Figure 2 are visually distinct, an algorithmic approach requires a quantitative score function to separate between the (vector-valued) conditional distributions generated from feature vector samples; and *(ii)* the global detector receives only finitely many samples; thus, we can construct (at best) only a noisy estimate of the conditional distribution.

We develop **ShapeScore** – a score function based on the Wasserstein distance [30] to resolve between conditional distributions. This distance has well-known robustness properties to finite-sample binning [31], [32] and is easily implementable for vectors, thus making it suitable for our setting. Given a collection of feature vector samples, we construct an empirical (vector) histogram of the FV samples, and determine the Wasserstein distance of this histogram with respect to a reference histogram. This reference histogram is constructed from the feature vectors corresponding to the *false positives* of the local detectors. In other words, this reference histogram captures the statistical shape of the "failures" of the LDs – i.e., those FVs that the LD classifies as malicious even though they arise from benign applications. For more details, see Section IV.

If we had the idealized scenario of infinite number of feature vector samples, the ShapeScore would be uniquely and deterministically known. In practice however, we have only a limited number of feature vector samples; thus ShapeScore is noisy. Figure 4 tests its robustness with Windows benign and malicious applications (see Section VI for details), where the ShapeScore is computed from neighborhood sizes of 15,000 FVs (about 15 seconds of data from 1000 nodes). The key observation is the strong statistical separation between the scores for the TP and FP feature vectors respectively, thus lending credence to our approach. Importantly, both these ideas do not depend on a fixed size of the neighborhood (or even knowing the neighborhood size); thus they provide another lens to study malware at a global level.

III. Related Work

A. *Behavioral analysis*

Behavioral analysis refers to statistical methods that monitor signals from program execution, extract features and build models from these signals, and then use these models to classify processes as malicious. Importantly, as we discuss in this section, all known behavioral detectors have a high false positive and negative rate (especially when zero-day and mimicry attacks are factored in).

System-calls and middleware API calls have been studied extensively as a signal for behavioral detectors [5], [7], [17], [33]–[36]. Network intrusion detection systems [2] analyze network traffic to detect known malicious or anomalous behaviors. More recently, behavioral detectors use signals such as power consumption [37], CPU utilization, memory footprint, and hardware performance counters [10], [11].

Detectors then extract *features* from these raw signals. For example, an n-gram is a contiguous sequence of n items that captures total order relations [5], [38], n-tuples are ordered events that do not require contiguity, and bags are simply histograms. These can be combined to create bags of tuples, tuples of bags, and tuples of n-grams [5], [33] often using principal component analysis to reduce dimensions. Further, system calls with their arguments form a dependency graph structure that can be compared to sub-graphs that represent malicious behaviors [17], [35], [39].

Finally, detectors train models to classify executions into malware/benignware using supervised (signature-based) or unsupervised (anomaly-based) learning. These models range from distance metrics, histogram comparison, hidden markov models (HMM), and neural networks (artificial neural networks, fuzzy neural networks, etc.), to more common classifiers such as kNN, one-class SVMs, decision trees, and ensembles thereof.

Such machine learning models, however, result in high false positives and negatives. Anomaly detectors can be circumvented by mimicry attacks where malware mimics system-calls of benign applications [34] or hides within the diversity of benign network traffic [3]. Sommer et al. [3] additionally highlights several problems that can arise due to overfitting a model to a non-representative training set, suggesting signature-based detectors as the primary choice for real deployments. Unfortunately, signature-based detectors cannot detect new (zero-day) attacks. On Android, both system calls [40] and hardware-counter based detectors [10] yield ∼20% false positives and ∼80% true positives.

B. Collaborative Intrusion Detection Systems (CIDS)

Collaborative intrusion detection systems (CIDS) provide an architecture where LDs' alerts are aggregated by a *global detector* (GD). GDs can use either signature-based or anomaly-based [4], [41], or even a combination of the two [42] to generate global alerts. Additionally, the CIDS architecture can be centralized, hierarchical, or distributed (using a peer-to-peer overlay network) [4].

In all cases, existing GDs use some variant of count-based algorithms to aggregate LDs' alerts [20], [21]: once the number of alerts exceeds a threshold within a space-time window, the GD raises an intrusion alert. In HIDE [4], the global detector at each hierarchical-tier is a neural network trained on network traffic information. Worminator [43] additionally uses bloom filters to compact LDs' outputs and schedules LDs to form groups in order to spread alert information quickly through a distributed system. All count based algorithms are fragile when the noise is high (in the early stages of an infection) and when the network size is uncertain. In contrast, our shape and censoring based GD is robust against such uncertainty.

Note that distributed CIDSs are vulnerable to probe-response attacks, where the attacker probes the network to find the location and defensive capabilities of an LD [44]–[46]. These attacks are orthogonal to our setting since we do not have fixed LDs (i.e. all nodes act as LDs).

Algorithm 1: Local Detector

```
   Input  : Real-time sequence of executed system calls
   Output: Alert-FVs
1  id – LD's identifier
2  while True do
3      $syscall\text{-}hist$ := $r$-sec histogram of system calls
4      $syscall\text{-}hist_{PCA}$ := project $syscall\text{-}hist$ on $L$-dim
        PCA basis
5      $label$ := BinaryClassifier($syscall\text{-}hist_{PCA}$);
6      if $label = malicious$ then
7          $alert\text{-}FV$ := $syscall\text{-}hist_{PCA}$
8          send $< alert\text{-}FV, id >$ to Shape GD
```

Algorithm 2: Neighborhoods from Attack-Vectors

```
   Input  : Template-type, NTW
   Output: Update the list of active neighborhoods
           NB-LIST
   [time, time+NTW] defines the current time window
1  time := current time
2  while True do
3      if Template-type = $waterhole\ attack$ then
4          V := client machines*
5          S := accessed servers*
6          predicate(A:Client, B:Servers) := A accesses B
7      else if Template-type = $phishing\ attack$ then
8          V := email recipient machines*
9          S := mailing lists*
10         predicate(A:Recipient, B:Mailing list) := A $\subseteq$ B
       partitioning a set into non-disjoint sets to incorporate
       structural filtering
11     $\bigcup_{i=1}^{N} P_i$ = partition-set(S)
       form neighborhoods $NB_i$ using partitions $P_i$
12     $NB_i$ = {V | predicate(V, $P_i$)}
       set expiration time for a neighborhood $NB_i$
13     $NB_i$.expiration-time = t+NTW
       add all neighborhoods to the list $NB\text{-}LIST$
14     $NB\text{-}LIST$ = {$NB_i$ | $\forall i$ in $[1, N]$}
       advance time by NTW sec
15     time = time+NTW
   *active within the time window [time, time+NTW]
```

IV. Shape GD Algorithm

Our algorithm consists of Feature Extraction (FE), Local Detector (LD), and the Global Detector (GD). Our key innovations are in the Global Detector, however, we describe each of these components below for notation and completeness.

Feature Extraction algorithm. This algorithm transforms the continuously evolving 390-dimensional time-series of Win-

Algorithm 3: Malware Detection in a Neighborhood

```
Input  : $L$-dim alert-FVs
Output: Malicious neighborhoods
1 $NB\text{-}LIST$ – list of neighborhoods
2 for each NB in NB-LIST do
3     $\mathcal{B}$ := {alert-FVs | node $id \subseteq$ NB}
4     $\mathcal{H}_{\mathcal{B}}$ := bin $\mathcal{B}$ along each dimension & normalize
5     $ShapeScore$ := Wasserstein Dist.($\mathcal{H}_{\mathcal{B}}$, $\mathcal{H}_{\text{ref}}$)
6     if $ShapeScore > \gamma$ then
7         label NB as malicious
```

dows system calls into a discrete-time sequence of feature-vectors (FVs). This is accomplished by chunking the continuous time series into $r-$second intervals, and representing the system call trace over each interval as a single $L-$dimensional vector (Algorithm 1, lines 3–4). L is typically a low dimension, in our experiments $L = 10$ and $r = 1$ second. This feature extraction can be accomplished, for instance, by using a histogram and PCA analysis (see Section V-C for details).

Local Detector (LD) Algorithm. The LD algorithm (Algorithm 1) leverages the current state-of-the-art techniques in automated malware detection to generate a sequence of *alerts* from the FV sequence. Specifically, using both its internal state and the *current* FV, the LD algorithm generates an alert if it thinks that this FV corresponds to malware, and produces no alert if it thinks that the current FV is benign (lines 5–7). Henceforth, we define an **alert-FV** to be an FV that generates an LD alert (either true or false positive). In our experiments each LD employs Random Forest as a binary classifier for malware detection because Random Forest demonstrates the best performance on the training data set (Figure 3).

Neighborhood Instances from Attack-Templates. Each neighborhood time window (NTW), Shape GD generates neighborhood instances (Algorithm 2) based on statically defined attack vectors – each attack vector is a “Template” to generate neighborhoods with. Algorithm 2 shows how the concept of neighborhood unifies operationally distinct attacks like waterhole and phishing.

The template for detecting a waterhole attack forms a neighborhood out of client nodes that access a server or a group of servers within a neighborhood time window (NTW). The other template, which is used for detection of a phishing attack, includes in a neighborhood email recipient machines belonging to a set of mailing lists. The two templates are shown in lines 4–11.

For simplicity we present a batch version of the neighborhood instantiation algorithm (Algorithm 2) which advances time by NTW and creates new neighborhoods for each NTW window. In contrast, the online Shape GD version updates already existing neighborhoods while monitoring client–server interactions in real-time – we demonstrate the online Shape GD algorithm to detect waterhole attacks and the batch version against phishing attacks in our evaluation.

The neighborhood instantiation algorithm accepts a template type as input, i.e. either a template for detecting a waterhole attack or a phishing attack, and length of a neighborhood time window (NTW). algorithm operates on time-window basis, where each time window spans NTW seconds. The algorithm starts by defining the sets V and S that are later used to form neighborhoods. For a waterhole attack, the set V includes all client machines accessing a set of servers and S is a set of the accessed servers. To instantiate neighborhoods for a phishing attack, V is a set of all email recipient machines and S is a set of mailing lists. In both cases the algorithm considers only the entities that are active within a current NTW window.

Each attack requires a predicate that determines relation between the elements of the sets V and S. For a waterhole attack such a predicate is true if a client *accesses* one of the servers (line 7). In the case of a phishing template, the predicate is evaluated to true if a recipient *belongs* to a particular mailing list (line 11).

The neighborhood instantiation algorithm proceeds with partitioning the set S into one or more disjoint subsets P_i (line 12). This is to incorporate ‘structural filtering’ into the algorithm, allowing an analyst to create neighborhoods based on subsets of servers (instead of all servers in case of waterhole) or divide all mailing lists into subsets of mailing lists (in the phishing). Structural filtering boosts detection under certain conditions (see Section VI-D).

The neighborhood instantiation algorithm builds a neighborhood for each partition P_i using a corresponding predicate (line 13). After forming a neighborhood, the algorithm sets its expiration time (line 14), which is the end of the current NTW window. All the neighborhoods in the list $NB\text{-}LIST$ are discarded at the end of the current NTW window. Finally, the algorithm adds the just formed neighborhoods to $NB\text{-}LIST$ (line 15) and advances time by one NTW (line 16).

The template-based neighborhood instantiation algorithm (Algorithms 2) shares the $NB\text{-}LIST$ data structure with the Algorithm 3 that uses neighborhoods’ shapes to detect malware.

Malware Detection in a Neighborhood. Algorithm 3) detects malware *per neighborhood* instead of individual nodes. The input to the algorithm is a set of alert-FVs from each neighborhood and its output is a global alert for the neighborhood. We now describe how the algorithm distinguishes between the *conditional distributions* of alert-FVs from true-positive and false positive neighborhoods.

The key algorithmic idea is to first construct a single histogram from all the alert-FVs from a neighborhood and compare this histogram with a reference histogram, *both from the same neighborhood*, to yield the neighborhood’s ShapeScore that is used for generating global alerts. The reference histogram is constructed from a set of *false positive* alert-FVs – thus, the reference histogram captures the statistical shape of misclassifications (FPs) by the LDs at a neighborhood scale.

Generating histograms from alert-FVs. The algorithm aggregates L-dimensional projections of alert-FVs on per neighborhood basis into a set $\mathcal{B}$ (Algorithm 3, line 3). After that, Shape GD converts low dimensional representation of alert-FVs, the set $\mathcal{B}$, into a single (L, b)-dimensional vector-histogram denoted by $\mathcal{H}_{\mathcal{B}}$ (line 4). The conversion is performed by binning L-dimensional vectors within the $\mathcal{B}$ set along each dimension. In each of the L-dimensions, the scalar-histogram of the corresponding component of the vectors is binned and normalized. Effectively, a vector-histogram is a matrix $L\text{x}b$, where L is the dimensionality of alert-FVs and b is the number of bins per dimension.

We use standard methods to determine the size and number of bins. In particular, we tried square-root choice, Rice rule, and Doanes formula [47] to estimate the number of bins, and we found that 20–100 bins yielded separable histograms (as in Figure 4) for the Windows dataset and fixed it at 50 for our experiments.

ShapeScore. We get the ShapeScore by comparing this histogram, $\mathcal{H}_\mathcal{B}$, to a *reference histogram*, denoted by $\mathcal{H}_{\mathrm{ref}}$. The reference histogram is generated using only the false positive FVs of the LDs. In other words, the local detectors are applied to the traces generated by benign apps; the FVs corresponding to the alerts from the LD (these are the false positives) are used to construct the reference histogram $\mathcal{H}_{\mathrm{ref}}$.

The ShapeScore of the accumulated set for FVs, $\mathcal{B}$, is given by the sum of the coordinate-wise Wasserstein distances [31] (Algorithm 3, line 5) between

$$\mathcal{H}_\mathcal{B} = (\mathcal{H}_\mathcal{B}(1)\ \mathcal{H}_\mathcal{B}(2)\ \ldots\ \mathcal{H}_\mathcal{B}(L))$$

and

$$\mathcal{H}_{\mathrm{ref}} = (\mathcal{H}_{\mathrm{ref}}(1)\ \mathcal{H}_{\mathrm{ref}}(2)\ \ldots\ \mathcal{H}_{\mathrm{ref}}(L)).$$

In other words,

$$\mathrm{ShapeScore} = \sum_{l=1}^{L} d_W(\mathcal{H}_\mathcal{B}(l), \mathcal{H}_{\mathrm{ref}}(l)),$$

where for two scalar distributions p, q, the Wasserstein distance [30], [31] is given by

$$d_W(p,q) = \sum_{i=1}^{b} \left| \sum_{j=1}^{i} (p(j) - q(j)) \right|.$$

This Wasserstein distance serves as an efficiently computable one dimensional projection, that gives us a discriminatively powerful metric of distance [31], [32]. Because the Wasserstein distance computes a metric between distributions – for us, histograms normalized to have total area equal to 1 – it is invariant to the number of samples that make up each histogram. Thus, unlike count-based algorithms, *it is robust to estimation errors in community size*. Figure 4 verifies this intuition, and shows that true positives and false positive feature vectors separate well when viewed through the ShapeScore.

Finally, to deploy the Shape GD, we determine a robustness threshold γ computed via standard confidence interval or cross-validation methods with multiple sets of false-positive FVs (see Section VI-A). If $ShapeScore > \gamma$, we declare a *global alert,* i.e., the algorithms predicts that there is malware in the neighborhood (lines 6–7).

V. Experimental Setup

A. Benign and Malware Applications

We collect data from thousands of benign applications and malware samples. To avoid tracing program executions where malware may not have executed any stage of its exploit or payload correctly, we set a threshold of 100 system calls per execution. In our experiments we were able to successfully run 1,889 out of 2,000 benign applications, 1,311 out of 2,000 malware samples from 193 malware families collected in July 2013 [48], and 2,364 out of 3,225 more recent samples from 13 popular malware families collected in 2015 [49].

We record time stamped sequences of executed system calls using Intel's Pin dynamic binary instrumentation tool. Each Amazon AWS virtual machine instance runs Windows Server 2008 R2 Base on the default T2 micro instances with 1GB RAM, 1 vCPU, and 50GB local storage. The VMs are populated with user data commonly found on a real host including PDFs, Word documents, photos, Firefox browser history, Thunderbird calendar entries and contacts, and social network credentials. To avoid interference between malware samples, we execute each sample in a fresh install of the reference VM. As malware may try to propagate over the local network, we set up a sub-net of VMs accessible from the VM that runs the malware sample. We left open common ports (HTTP, HTTPS, SMTP, DNS, Telnet, and IRC) used by malware for communication with its command and control servers (C&C). We run each benign and malware program 10 times for 5 minutes per run for a total of almost 53,000 hours total compute time on Amazon AWS.

Overall, benignware and malware were active for 141,670 sec and 283,270 seconds respectively, executing an average of 11,900 and 13,500 system calls per second respectively. Using 1 second time window (Section IV) and sliding the time windows by 1ms, we extract histograms of system calls within each time window as the ML feature, and finally pick 1.5M benign and 1M malicious FVs from this dataset for the experiments that follow.

B. Modeling Waterhole and Phishing Attacks

Waterhole attack. To model a waterhole attack, we use Yahoo's "G4: Network Flows Data" [50] dataset, which contains communication data between end-users and Yahoo servers. The 41.4 GB (in compressed form) of data were collected on April 29-30, 2008. Each netflow record includes a timestamp, source/destination IP address, source/destination port, protocol, number of packets and the number of bytes transferred from the source to the destination [1].

Specifically, we use 5 hours of network traffic (208 million records) captured on April 29, 2008 between 8 am and 1 pm at the border routers connecting Dallas Yahoo data center (DAX) to the large Internet. We further restrict the simulated network to include only 50 most frequently accessed DAX servers and 3,181,127 client machines, which initiate 14,249,931 requests total.

[1]All IP addresses in the dataset are anonymized using a random permutation algorithm, thus it is impossible to trace them back to the real servers

We assume that an attacker compromises one of the most frequently accessed DAX server – 118.242.107.76, which processes $\sim 752,000$ requests within 5-hour time window (~ 43.7 requests per second). In our simulation it gets compromised at random instant between 8am and 10.30am. Hence, Shape GD can use the remaining 2.5 hours to detect the attack (our results show that less than a hundred seconds suffice). Following infection, we simulate this 'waterhole' server compromising client machines over time with an infection probability parameter – this helps us determine the time to detection at different rates of infection. The benign and compromised machines then select corresponding type of FVs (generated in Section V-A) and input these to their LDs.

Phishing attack. We simulate a phishing attack in a medium size corporate network of 1086 nodes that exchange emails with others in the network. To model email communication, we pick 50 email threads with 100 recipients each from the publicly available Enron email dataset [51] (the union of all email threads' recipients is 1086).

We start the simulation with these 50 emails being sent into the 1086-node neighborhood, and seed only *one* email out of 50 as malicious. We then model the infection speading at different rates as this malicious email is opened by its (up to 100) recipients at some time into the simulation and is compromised with some likelihood when the user 'clicks' on the URL in the email. Our goal is to measure the number of compromised nodes before Shape GD declares an infection in this neighborhood. All nodes that open and 'click' the link in the malicious email will select malware FVs from Section V-A as input to their corresponding LDs, while the remaining nodes select benign FVs.

To simulate the infection spreading over the email network, we need to (a) model when a recipient 'opens' the email: we do so using a long tail distribution of reply times where the median open time is 47 minutes, 90-percentile is one day, and the most likely open time is 2 minutes [52]; and (b) model the 'click' rate (probability that a recipient clicks on a URL): we vary it from 0% up to 100% to control the rate of infection. For example, within 1-, 2-, 3-hour long time interval only 55%, 65%, and 70% of recipients of a malicious email open it, which corresponds to 55, 65, and 75 infected machines respectively at 100% click rate.

Overall, these two scenarios differ in their time-scales (seconds v. hours) and in the relative rate at which benign and malicious neighborhoods grow. As we will see, these parameters have a significant impact on the composition of neighborhoods and the Shape GD's detection rate.

Methodology. We report averaged results from repeating each experiment multiple times with random initialization parameters. In particular, we use 10-fold cross validation for machine learning experiments (Figure 3), 500 randomly sampled benign/malicious neighborhoods with 10 repetitions to compute average (Figures 4, 5), 100 repetitions of each malware infection experiment (Figures 6,7,12,13), and 100 repetitions of infection with 10 repetitions per data-point (Figures 8,9,10,11).

All Shape GD's parameters are chosen based on a training data set, which is completely separate from the testing partition used for evaluation purposes. Moreover, the data used for the experiments in Section VI-A, which guided the choice of Shape GD's parameters, is not reused for Shape GD's performance results.

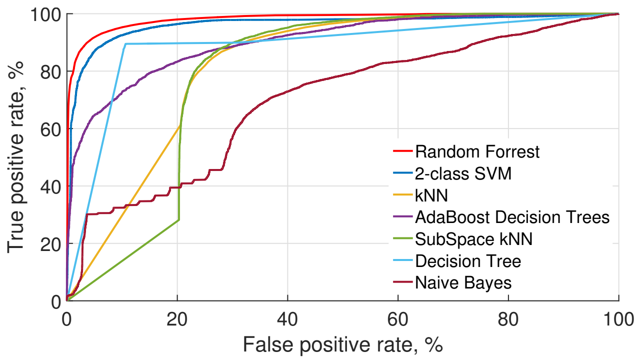

Fig. 3. (ROC curves) True positive v. False positive curves shows detection accuracy of seven local detectors. Random Forest outperforms all others; but has unacceptably high false positive rate (above 10%) if one wants to achieve at least 95% true positive rate.

C. Local Detectors

Our first step is to establish a good local detector (LD) for desktop systems running Windows OS. In particular, we choose system call based LDs since the system call interface has visibility into an app's intercation with core OS components – file system, Windows registry, network – and can thus capture signals relevant to malware executions.

We experiment with an extensive set of system-call LDs – our takeaway is that even the best LD we could construct operates at a true- and false-positive ratio of 92.4%:6% and is not deployable by itself (i.e., will create ∼30 false positives every 10 minutes without a GD).

Each LD comprises of a feature extraction (FE) algorithm and a machine learning (ML) classifier. Our FE algorithm partitions the time-series of system calls into 1-sec chunks and represents each chunk as a histogram (where each bin contains frequency of a particular system call). Then it projects all feature vectors onto 10-dimensional subspace spanned by top 10 principal components generated by PCA algorithm. We choose ML classifiers (used throughout prior work because these are computationally efficient to train) such as SVMs, random forest, k-Nearest Neighbors, etc, and do not include complex alternatives such as artificial neural networks or deep learning algorithms. We also deliberately avoid handcrafted ML algorithms and hardcoded detection rules.

Figure 3 plots the true positive v. false positive rates (i.e. the ROC curves) of the seven ML algorithms we evaluate. The area under the ROC curve (AUC) is a quantitative measure of LD's performance: the larger the AUC, the more accurate the detector. We specifically experiment with seven state-of-the-art ML algorithms: random forest, 2-class SVM, kNN, decision trees, naive Bayes, and their ensemble versions – boosted decision trees with AdaBoost algorithm and Random SubSpace ensemble of kNN classifiers (Figure 3). We also evaluated 1-class SVM as an anomaly detector – however, it yielded an extremely high FP rate and we exclude it from further discussion. Overall, the random forest classifier worked best – it has the highest AUC and yields 92.4% true positives at a false positive rate of 6%.

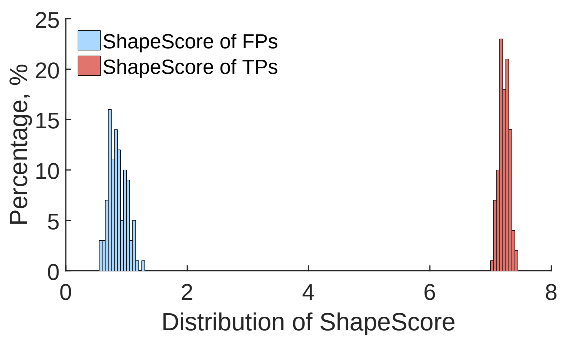

Fig. 4. Histogram of the ShapeScore: The ShapeScore is computed from neighborhood sizes of 15,000 FVs (experiment repeated 500 times to generate the histograms). Shape-based GD can reliably separate FPs and TPs through extracting information from the data that has been unutilized by an LD.

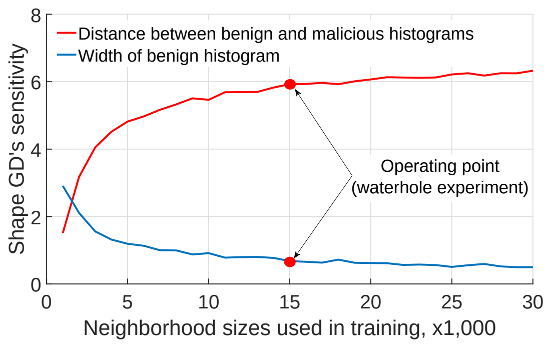

Fig. 5. Analysis of ShapeScore histogram parameters when changing neighborhood size. The curves flatten out on the right side from the operating point.

VI. Results

In this section, we quantify Shape GD's early and robust detection of infection as well as the baseline Count GD's fragility to small errors in estimating neighborhood size. In particular, we find that the shape of a neighborhood – i.e. the distribution of alert-FVs – can identify malicious neighborhoods with less than 1% false positive and 100% true positive rate using only 15,000 FVs[2] We then simulate realistic attack scenarios and find that Shape GD can detect malware when only 22 of 1086 nodes are infected through phishing in an enterprise email network, and when *less than 150* of 550K possible nodes are infected through a waterhole attack using a popular web-service.

A. Can shape of alert-FVs identify malicious neighborhoods?

We first show that the shape of a neighborhood can easily distinguish between neighborhoods that are either 100% benign or 100% malicious. In real systems under, for example, phishing or waterhole attacks, the Shape GD has to operate under harsher conditions – detecting malware infections over neighborhoods with a small fraction of infected nodes – and we quantify Shape GD's time to detection in subsequent sections.

[2]Recall that at 60 FVs/node/minute, it takes 1000 nodes only 15 seconds to create 15,000 FVs. For LDs with ∼6% false positive rate, this corresponds to 900 alert-FVs.

Figure 4 shows that Shape GD can indeed, given a sufficient number of FVs, separate purely benign neighborhoods from purely malicious ones. To conduct this experiment, we construct benign and malicious neighborhoods by sampling FVs uniformly at random from the set of all benign and malicious FVs respectively. For this experiment, we set each neighborhood to be 15,000 FVs – in the next experiment, we will justify this size and evaluate the Shape GD's sensitivity to neighborhood size.

We then process each FV through an LD – the Random Forest LD trained on both benign and malicious system call histograms – to obtain a set of 'alerts'. An alert is either a false positive (FP) or a true positive (TP). The Shape GD receives the set of *alert-FVs* (an FV that led to an LD alert) per neighborhood and computes the ShapeScore (see Section IV). Recall that a small ShapeScore indicates the neighborhood's statistical shape is similar to that of a benign one.

In Figure 4, we compute the ShapeScore with 15,000 FVs (filtered through an LD with about 6% FP rate and 92.4% TP rate), and repeat this experiment 500 times for both benign and malicious FVs. The histograms of the results are plotted, where each point in the blue (or red) histogram represents a benign (or malicious) ShapeScore. The non-overlapping distributions separated by a large gap indicate that the shape of purely benign neighborhoods is very different from the shape of purely malicious neighborhoods

The Shape GD detects anomalous neighborhoods by setting a threshold score based on the distribution of benign neighborhoods' scores (Figure 4) – if an incoming neighborhood has a score above the threshold, Shape GD labels it as 'malicious', otherwise 'benign'. We set the threshold score at 99-percentile (i.e. our expected *global false positive rate* is 1%) and the true positive rate is effectively 100% for this experiment. This shows that for homogeneous neighborhoods over 15K FVs, Shape GD can make robust predictions.

B. How many FVs does Shape GD need to make robust predictions?

Neighborhood size is a crucial parameter for a Shape GD. With too few FVs, benign neighborhoods shape will have high variance (i.e. benign distribution in Figure 4 becomes wide and the gap between two distributions shrinks), leading to false negatives and positives. On the other hand, if neighborhoods are large, their shapes will be dominated by the large number of benign FVs and thus lead to missed alerts (false negatives) especially in the early stages of infection. Further, the number of alert-FVs per neighborhood in a deployed Shape GD need to be comparable to (or larger than) those used in training – due to all these reasons, we want to find the smallest number of FVs Shape GD needs to make robust predictions.

Figure 5 shows the sensitivity of Shape GD to neighborhood size (i.e., the number of FVs in a neighborhood during training stage). We vary the size of neighborhoods used in training from 3,000 up to 30,000 FVs and average the result of 10 experiments. We present two metrics in Figure 5 – the red curve plots the inter-class distance (between histograms of benign and malicious neighborhoods from Figure 4), and the blue curve plots intra-class distance (i.e. the width of the benign histogram). Figure 5 shows that the red inter-class

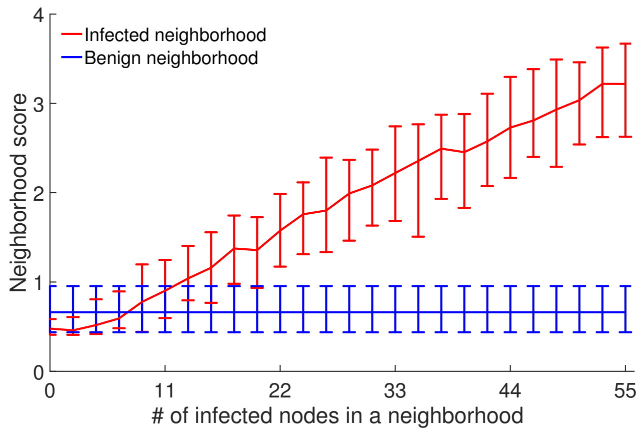

Fig. 6. (Phishing attack: Time-based NF) Dynamics of an attack: While the portion of infected nodes in a neighborhood increases over time reaching 55 nodes out of 1086 on average, ShapeScore goes up showing that Shape GD becomes more confident in labeling neighborhoods as 'malicious'. It starts detecting malware with at most 1% false positive rate when it compromises roughly 22 nodes. The neighborhood includes all 1086 nodes in a network and spans over 1 hour time interval.

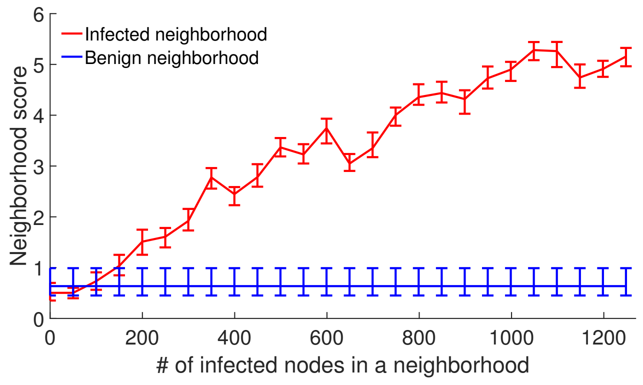

Fig. 7. (Waterhole attack: Time-based NF) Dynamics of an attack: While the portion of infected nodes in a neighborhood increases over time reaching 1248 nodes on average, ShapeScore goes up showing that Shape GD becomes more confident in labeling neighborhoods as 'malicious'. It starts detecting malware with at most 1% false positive rate when roughly 200 nodes get compromised. The neighborhood includes 17178 nodes on average and spans over 30 sec time interval.

distance increases (and blue intra-class variance decreases) quickly as neighborhood size increases, and both curves flatten out once the neighborhoods are approximately 15,000 FVs large.

This shows that (for our Windows programs dataset) neighborhoods with 15,000 FVs or more are a good choice to train Shape GD because purely malicious or benign distributions stabilize at this size. In real scenarios with mixtures of mostly benign and a few malicious neighborhoods, the number of FVs will have to be scaled up depending upon the timescale of attacks (hours for phishing v. seconds for waterhole) and the number of nodes affected by an attack (100s or 1000s in enterprise networks v. 100s of thousands in a broader waterhole attack). In the phishing and waterhole attack case studies that follow, we use neighborhoods of 100K FVs and 15K FVs respectively.

C. How early can Shape GD detect malware infections using temporal neighborhoods?

Temporal filtering creates a neighborhood using only the nodes that are *active* within a neighborhood time window (NTW). For example, a temporal neighborhood for the phishing scenario would include every email address that received an email within the last hour (1086 nodes in our experiments). Similarly, a waterhole attack scenario would include all client devices that accessed *any* server within the last NTW into one neighborhood ($\sim 17,000$ nodes on average in 30 seconds). This neighborhood filtering models a CIDS designed to detect malware whose infection exhibits temporal locality (and obviously does not detect attacks that target a few high-value nodes through temporally uncorrelated vectors).

Phishing and waterhole attacks operate at different time scales (and hence NTWs). Due to the long tail distribution of email 'open' times, the phishing NTW varies from 1–3 hours in our experiments. On the other hand, a popular waterhole server quickly infects a large number of clients within a short period of time – thus, we vary the waterhole NTW from 4 seconds up to 100 seconds.

Shape GD's time to detection for one NTW. We fix NTWs (1 hour for phishing and 30 seconds for waterhole) and vary a parameter that represents a node's likelihood of infection from 0% up to 100% – modeling whether a phished user clicks the malicious URL (phishing) or a drive-by exploit succeeeds in a waterhole attacks.

Figures 6 and 7 plot the neighborhood score v. the average number of infected nodes within benign (blue curve) and malicious (red curve) neighborhoods – the two extreme points on the X-axis corresponds to either none of the machines being infected (the left side of a figure) or the maximum possible number of machines being infected (the right side of the figure). In this experiment, phishing can infect up to 55 machines in the 1 hour NTW, while the waterhole server can infect almost 1250 nodes in the 30 seconds NTW. Every point on a line is the median neighborhood score from 10 experiments with whiskers set at 1%- and 99%- percentile scores.

When increasing the number of infected nodes in a neighborhood, as expected, the red curve larger deviates from the blue one. Therefore, Shape GD becomes more confident with labeling incoming partially infected neighborhoods as malicious. Shape GD starts reliably detecting malware very quickly – when only 22 nodes (phishing) and 200 nodes (waterhole) have been infected. We also experimented with other sizes of neighborhood window – the plots we obtained showed similar trends.

Shape GD's sensitivity to NTW. We show that the size of a neighborhood significantly affects early detection – the minimum number of nodes that are infected before Shape GD raises an alert – in Figures 8 and 9. Varying the NTW essentially competes the rates at which both malicious and benign FVs accumulate – interestingly, we find that *these relative rates are different for phishing and waterhole attacks and lead to different trends for detection performance v. NTW.*

We vary the NTW from 1 hour to 3 hours for phishing and from 4 sec to 100 sec for waterhole and record the number of

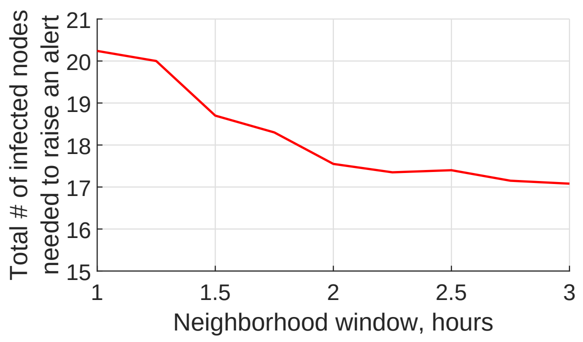

Fig. 8. (Phishing attack: Time-based NF algorithm) Shape GD's performance improves by 18.5% (20.24 and 17.08 infected nodes) when increasing the size of a neighborhood window from 1 hour to 3 hours.

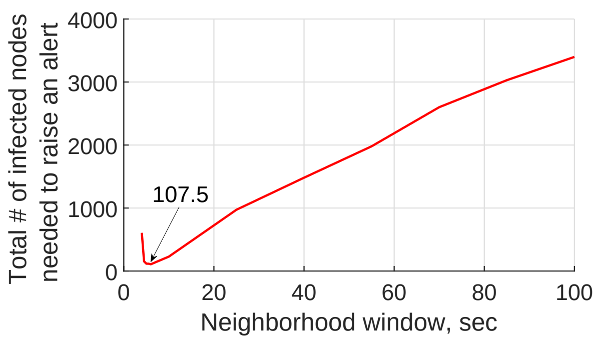

Fig. 9. (**Waterhole attack: time-based NF**) Shape GD's performance deteriorates linearly when increasing the size of a neighborhood window from 6 sec to 100 sec.

infected nodes when Shape GD can make robust predictions (i.e. less than 1% FP for almost 100% TP).

Increasing the NTW in the phishing experiment from 1 to 3 hours *improves* the Shape GD's detection performance – at 17.08 infected nodes for a 3 hour NTW compared to 20.24 nodes for a 1 hour NTW. Detection improves slightly because while the infection rate slows down over time as fewer emails remain to be opened, the long tail distribution of email 'open' times causes most of the 17 victims to fall early in the NTW and accumulate sufficient malicious FVs to tip the overall neighborhood's shape into malicious category. If the neighborhood was larger or the 'open' time distribution was not skewed, a larger NTW would include more benign FVs, and the trend in Figure 8 would reverse (as in the waterhole example that follows).

In a waterhole scenario, the number of client devices active within a time window (and hence the fraction of benign nodes and false positives from a neighborhood) grows much faster than the malware can spread (even when we assume that *every* client that visits the waterhole server gets infected, i.e. an infection rate of 100%). That is, a large NTW aggregates many more benign (false positive) FVs from clients accessing non-compromised servers. Hence, compared to the phishing attack, we observe an opposite trend – increasing the NTW degrades time to detection. The Shape GD works best with an NTW of 6 seconds – only 107.5 nodes on average become infected out of a possible $\sim 550,000$ nodes. A very small NTW (below 6 seconds) either does not accumulate enough FVs for analysis – if so, Shape GD outputs no results – or creates large variance in the shape of benign neighborhoods and abruptly degrades detection performance.

Note that a Shape GD requires a minimum number of FVs per neighborhood to make robust predictions – at least 15,000 FVs based on Section VI-B – hence, the Shape GD has to set NTWs based on the rate of incoming requests and access frequency of a particular server. For example, if a server is not very popular and is likely to be compromised, the Shape GD could increase this server's NTW to collect more FVs for its neighborhood.

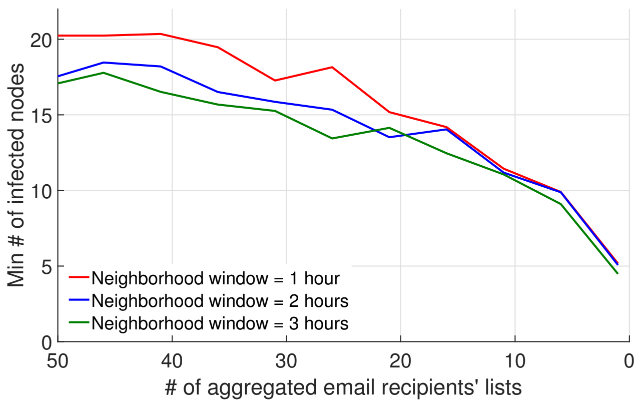

Fig. 10. (Phishing attack) Comparing to pure time-based NF, structural filtering algorithm improves Shape GD's performance by $\sim 4\times$ by taking into consideration logical structure of electronic communication (sender – receiver relation).

D. Can neighborhood structural information improve Shape GD's time to detection?

Both phishing and waterhole attacks impose a logical structure on nodes (beyond their time of infection): phishing spreads malware through malicious email attachments or links while waterhole attacks infect only the clients that access a compromised server. This structure suggests that temporal neighborhoods can be further refined based on the sender/recipient-list of an email (e.g., grouping members of a mailing list into a neighborhood in the phishing scenario) or based on the specific server accessed by a client (i.e., grouping clients that visit a server into one neighborhood).

To analyze the effect of such structural filtering on GD's performance, we vary filtering from coarse- (no structural filtering, only time-based filtering) to fine-grained (aggregating alerts across each recipients' list separately or across clients accessing each server separately) (Figures 10, 11). Specifically, the aggregation parameter changes from 50 recipients' lists or servers down to 1. As before, we measure detection in terms of the minimum number of infected nodes that lead to raising a global alert. Also we consider three NTW values – 1-, 2-, and 3- hours long for phishing and 25-, 50-, and 100-sec long for waterhole.

Figure 10 shows that structural filtering improves detection of a phishing attack by $\sim 4x$ (difference between left and

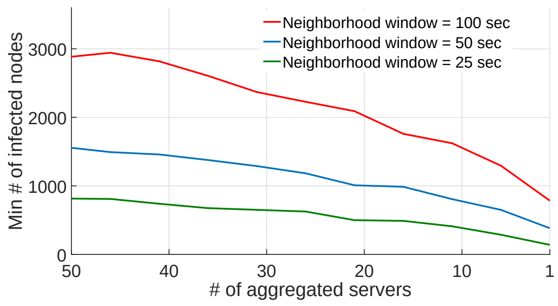

Fig. 11. (Waterhole attack) Comparing to pure time-based NF, structural filtering algorithm improves Shape GD's performance by $3.75\times$ – $5.8\times$ by aggregating alerts on a server basis.

right end points of each curve) over temporal filtering – by filtering out alert-FVs from unrelated benign nodes that were active during the same NTW as infected nodes. Interestingly, the size of a neighborhood window does not considerably affect the detection when used along with the most fine-grained structural filtering (treating each recipients' list individually) – 3-hour long NTWs results in only a $\sim$ 12% decrease in the number of compromised nodes (i.e. time to detection). This shows that there is substantial signal that structural filtering can help extract from alert-FVs in smaller NTWs (and thus improve Shape GD's time to detection).

Structural filtering improves time to detect waterhole attacks as well – by 5.82x, 4.07x, and 3.75x for 25-, 50-, 100-sec long windows respectively. Interestingly, structural filtering requires Shape GD to use longer NTWs than before – small NTWs (such as 6 seconds from the last sub-section) no longer supply a sufficient number of alert-FVs for Shape GD to operate robustly. Even though structural filtering with a 25 second NTW improves detection by 5.82x over temporal filtering with 25 second NTWs, the number of infected nodes at detection time is 139.9 – higher than the 107 infected nodes for temporal filtering with a *6 second* NTW (Figure 9). Temporal and structural filtering thus present different trade-offs between detection time and work performed by GD – their relative performance is affected by the rate at which true and false positive FVs are generated.

E. How fragile is state-of-the-art Count GD to errors in estimating neighborhood size?

A Count GD algorithm counts the number of alerts over a neighborhood and compares to a threshold to detect malware. As discussed earlier, this threshold should scale linearly in the size of the neighborhood, i.e. the total number of FVs generated within a neighborhood. Thus it is crucial for Count GD to be able to estimate neighborhood size precisely.

We now experimentally quantify the error Count GD can tolerate in phishing (Figure 12) and waterhole (Figure 13) settings. Note that the error in estimating neighborhood size can be double sided – underestimates (negative error) can make neighborhoods look like alert hotspots and lead to false positives, while overestimates (positive error) can lead to missed detections (i.e., lower true positives).

We run Count GD in the same setting as Shape GD when evaluating time-based NF (Section VI-C) – 1-hour long

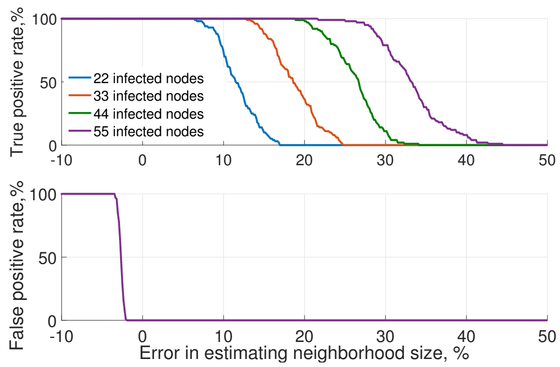

Fig. 12. (Phishing attack) An error in estimating neighborhood size dramatically affects Count GD's performance. It can tolerate at most 2% underestimation errors and 6.3% overestimation errors to achieve comparable with Shape GD performance.

neighborhood time window (NTW) with 1086 nodes (Figure 12) to model phishing and 30-sec long neighborhood including 17,178 nodes (Figure 13) to model a waterhole attack. We vary click rate in emails (phishing) and infection probability (waterhole) such that the number of infected nodes in a neighborhood changes from 0 to 55 (phishing) and from 0 to 500 (waterhole) in four increments – note that in both scenarios, only a small fraction (5.5% and 2.9%) of nodes per neighborhood get infected in the worst case.

In this setting, recall that the Shape GD has a maximum global false positive rate of 1% and a true positive rate of 100% – and detects malware when only 22 (phishing) and 200 (waterhole) nodes are infected – for the same NTWs. When the same number of nodes are infected, and for a similar detection performance, our experiments show that the Count GD can only tolerate neighborhood size estimation errors within a very narrow range – [-2%, 6.3%] (phishing) and [-0.1%, 13.8%] (waterhole). A key takeaway here is that underestimating a neighborhood's size makes Count GD extremely fragile (-2% in phishing and -0.1% for waterhole). On the other hand, overestimating neighborhood sizes decreases true positives, and this effect is catastrophic by the time the size estimates err by 17% (phishing) and 17.5% (waterhole).

We comment that this effect can be important in practice. Given the practical deployments where nodes get infected out of band (e.g., outside the corporate network), go out of range (with mobile devices), or with dynamically defined neighborhoods based on actions that can be missed (e.g. neighborhood defined by nodes that 'open' an email instead of only downloading it from a mail server), the tight margins on errors can render Count GD extremely unreliable. Even with sophisticated size estimation algorithms, recall that the underlying distributions that create these neighborhoods (email open times, number of clients per server, etc) have sub-exponential heavy tails [52] – such distributions typically result in poor parameter estimates due to lack of higher moments, and thus, poorer statistical concentrations of estimates about the true value [53]. Circling back, we see that by eliminating this size dependence compared to Count GD, our Shape GD provides a robust inference algorithm.

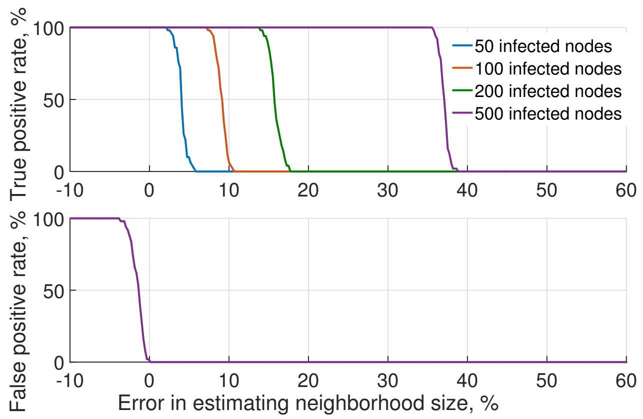

Fig. 13. (Waterhole attack) An error in estimating neighborhood size dramatically affects Count GD's performance. It can tolerate at most 0.1% underestimation errors and 13.8% overestimation errors to achieve comparable with Shape GD performance.

VII. Deployment

Implementation. Shape GD is meant to serve as an additional enterprise-level network protection mechanism. Currently, enterprises use SIEM tools (like HP Arcsight) to monitor network traffic and collect network logs, and email monitoring and virus scanning tools, in addition to host-based malware detectors (LDs) from McAfee, Lookout, etc. We use exactly these logs (client-IP, server IP, timestamp) data to construct neighborhoods and thus filter alert-FVs from the LD nodes (Algorithm 2).

Upon receiving alert-FVs, Shape GD runs its malware detection algorithm (Algorithm 3) for all neighborhoods that those alert-FVs belong to. If a particular neighborhood is suspicious, then Shape GD will notify a higher level protection system and forward any relevant information as an incident report. This higher level protection system may include deep static/dynamic code analysis and/or involve human analysts.

Computing Shape GD's parameters. Here we elaborate on the steps that should be taken in a real world environment to choose parameters. The steps discussed here are generic and are applicable to other attacks beyond waterhole and phishing.

First, an analyst should start with designing an appropriate algorithm to run on local detectors (LDs) (Section V-C). To achieve this, an analyst needs to compare the performance of multiple feature extraction (FE) algorithms combined with a diverse set of machine learning classifiers. One way to choose the best pair of a FE algorithm and a classifier is to build ROC curves for each pair, and select the pair that is closest to the desired operating point (the operating point determines the false positive and true positive rates for a local detector).

Second, the analyst needs to determine the minimum number of FVs that should be generated within a neighborhood for robust detection of infected neighborhoods (Section VI-B). This number depends on the false positive rate of LDs (e.g. in our experiments, we determined that a neighborhood should generate at least 15K FVs, see Figure 5).

Third, we need to choose an NTW based on the false positive rate (FP) and the desired time-to-detection. A small NTW means more frequent transfers of FVs from LD to GD, whereas a long NTW means that more nodes can get compromised before the GD makes a decision and/or FPs can drown out TPs.

Reference histogram. As explained in the Section IV, a reference histogram lies at the heart of Shape GD. To construct it, we need to obtain the LD's false positive vectors (FPs). A straightforward approach is to collect FPs in a lab environment using test inputs on benign apps in a malware-free system.

VIII. Overhead

Recall that Shape GD requires only the low-dimensional alert FVs – this leads to a two-fold dimensionality reduction when sending data from individual nodes to the GD. First, the FVs are low-dimensional (here, 10-dimensional vectors). Second, only alert FVs are needed – this leads to a 16-fold reduction in data (roughly only 6% of the FVs lead to alerts). Further, the Shape GD is a batch processing algorithm, thus, the individual nodes can batch their data at coarse time-scales (e.g. once every NTW) and send the data to the Shape GD. Finally, it does not matter even if some batches are lost/missed; recall that the Shape GD is robust to precisely this type of noise. Now we delve into overhead associated with individual components of our system.

Local detectors. Generating a single FV, which is a 1-sec histogram of system calls, on a local host is equivalent to performing 2,500 (system call frequency) direct table lookups on average and incrementing corresponding counters. Projection on a PCA basis requires to compute 10 dot products. Finally, running an LD, which is Random Forest in our case, results in performing 330 scalar comparisons on average.

Data transfer. Each FV is composed of 10 floating point numbers (40 bytes total if assuming single precision format). In the phishing experiment 1086 hosts transfer (in aggregate) $\sim 40KB/sec$; data transfer rate in waterhole setting is a little bit higher: $\sim 4,450$ hosts transfer (in aggregate) $\sim 174KB/sec$. In both cases we assume Shape GD using pure time-based filtering with 1 hour and 6 sec neighborhood time windows respectively.

If Shape GD employs structural filtering on top of the time-based one, then data transfer depends on the number of emails floating in a network or on the number of servers. In both cases, data transfer scales linearly with the number of emails and servers. When applying the most fine-grained structural filtering in our experiments, the nodes susceptible to phishing attacks transfer $\sim 4KB/sec$ per email and the nodes susceptible to waterhole attacks send $\sim 40KB/sec$ per server when using 1 hour and 25 sec neighborhood windows respectively.

Server computations. After receiving a batch of alert-FVs, Shape GD performs lightweight computations. Overhead of binning scales linearly with the number of alert-FVs in a batch; each binning operation is a direct table lookup together with counter increment. Calculating ShapeScore, which is Wasserstein distance, results in a sequence of additive operations, whose total number is equal to the dimensionality of FVs, which is 10, multiplied by the number of bins, which is 50.

IX. Discussion

Global FPs vs LD FPs. As remarked in the Introduction, an FP of 1% at the global level means that we will see one alert every 100 - 300 hours (for the phishing scenario) and 100 seconds (for waterhole scenario the neighborhood time window slides by 1 second). This reduces work to be performed by the deeper, second-level analysis considerably.

Specifically, LDs operating at 6% false positive rate generate 23.5M – 70M FPs within 100–300 hours time interval in a network of 1086 nodes (phishing) and 300K alerts within every 100 sec interval where neighborhoods include $\sim$50K nodes on average (waterhole). Shape GD filters these alerts. When using 1–3 hours (phishing) and 6 sec (waterhole) time-based neighborhood filtering, Shape GD will report to a system running a deeper analysis approximately 234.5K – 703.7K FPs raised by LDs (phishing) and approximately 1.4K FPs (waterhole). Adding structural filtering brings these numbers down to 21.6K – 64.8K FPs (phishing) and 360 FPs (waterhole).

Compared to a neighborhood of LDs, Shape GD thus reduces the number of FPs reported to deeper analyses by $\sim 100\times$ and $\sim 200\times$ when employing time-based filtering only (for phishing and waterhole scenarios respectively), while structural filtering reduces alert-FVs for deeper analysis to $\sim 1000\times$ and $\sim 830\times$. In both scenarios, analysts can choose to reduce number of alert-FVs to be analyzed by sliding neighborhood windows by a larger interval; however, this will increase the time to detect malware infection.

Heterogeneous neighborhoods. We have evaluated Shape GD for the case of homogeneous neighborhoods, i.e. all the nodes run the same operating system (Windows) and LDs of the same type (Random Forest). The shape idea may apply to heterogeneous neighborhoods comprising of heterogeneous devices (e.g. mobile, desktop), and with heterogeneity among the LDs (e.g. operating at different levels of the hardware/software stack, different versions).

An extension of our work to this setting is plausible by constructing a vector-histogram (Section IV) of higher dimensionality (in the worst case, number of LD types times the number of system types). This would potentially work if there are sufficient numbers of nodes of each type, so that the higher dimension vector-histogram exhibits ‘typical’ statistics. Such a setting might be particularly interesting for security providers who are monitoring large networks, where there would be a sufficiency of nodes in a neighborhood for such statistical concentrations to occur. Finally, a particularly interesting scenario would be to deploy multiple lightweight heterogeneous LDs on each system and perform intra-system statistical shape analysis. We leave these directions for future work.

Deep Learning. Behavioral analysis using machine learning approaches are becoming increasingly popular in industry, thanks to the vast amount of data and new “high dimensional” algorithms that are becoming prevalent (e.g. deep nets). With their ability to extract highly effective features, deep nets may provide a new way forward for creating novel behavioral detectors. At the global level, however, what is needed is a data-light approach for global detection by composing local detectors, tailored to be agile enough to do global detection in a fast-changing (non-stationary) environment.

Our work addresses precisely this problem. We believe that our approach effectively serves as a lens for dimensionality reduction and is thus complementary to the state-of-the-art behavioral analyses approaches.

X. Conclusions

Building robust behavioral detectors is a long-standing problem. This paper makes significant progress towards this goal by dealing with variability in community-membership and user-notifications. Our key ideas are to first use local detectors as a filter for feature vectors per neighborhood – instead of counting LDs’ alerts – and then to observe that the conditional distribution of feature vectors (i.e. after filtering by LDs) can separate malicious neighborhoods from benign ones. Interestingly, this property does not depend on community size variations. We show that this can reduce time to detection and global false positives/negatives significantly. This shape is a new property – future work will focus on leveraging this by co-designing *weaker* detectors to gain energy-performance efficiency on client devices and relying on the global detector to recoup accuracy.

References

[1] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, “Bothunter: Detecting malware infection through ids-driven dialog correlation,” in *Proceedings of 16th USENIX Security Symposium*, 2007.

[2] V. Paxson, “Bro: A system for detecting network intruders in real-time,” *Comput. Netw.*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[3] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” in *the IEEE Symposium on Security and Privacy*, 2010.

[4] Z. Zhang, J. Li, C. N. Manikopoulos, J. Jorgenson, and J. Ucles, “Hide: a hierarchical network intrusion detection system using statistical preprocessing and neural network classification,” in *the IEEE Workshop on Information Assurance and Security*, 2001.

[5] D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda, “A quantitative study of accuracy in system call-based malware detection,” in *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, ser. ISSTA 2012, 2012.

[6] S. J. Mihai Christodorescu, “Static analysis of executables to detect malicious patterns,” The University of Wisconsin, Madison, Tech. Rep., 2006.

[7] M. Fredrikson, S. Jha, M. Christodorescu, R. Sailer, and X. Yan, “Synthesizing near-optimal malware specifications from suspicious behaviors,” in *IEEE Symposium on Security and Privacy*, 2010.

[8] S. Hanna, L. Huang, E. Wu, S. Li, C. Chen, and D. Song, “Juxtapp: A scalable system for detecting code reuse among android applications,” in *Detection of Intrusions and Malware, and Vulnerability Assessment*, 2013.

[9] M. Christodorescu, S. Jha, S. A. Seshia, D. Song, and R. E. Bryant, “Semantics-aware malware detection,” in *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, 2005. [Online]. Available: http://dx.doi.org/10.1109/SP.2005.20

[10] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, “On the feasibility of online malware detection with performance counters,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, 2013.

[11] A. Tang, S. Sethumadhavan, and S. Stolfo, “Unsupervised anomaly-based malware detection using hardware features,” in *Research in Attacks, Intrusions and Defenses*, 2014.

[12] K. N. Khasawneh, M. Ozsoy, C. Donovick, N. Abu-Ghazaleh, and D. Ponomarev, “Ensemble learning for low-level hardware-supported malware detection,” in *Research in Attacks, Intrusions, and Defenses*. Springer International Publishing, 2015, pp. 3–25.

[13] J. Z. Kolter and M. A. Maloof, “Learning to detect and classify malicious executables in the wild,” *J. Mach. Learn. Res.*, vol. 7, pp. 2721–2744, Dec. 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1248547.1248646

[14] “Cisco annual security report 2014,” http://www.cisco.com/web/offers/lp/2014-annual-security-report/index.html, Tech. Rep., 2013.

[15] W. Xu, Y. Qi, and D. Evans, “Automatically evading classifiers: A case study on pdf malware classifiers,” in *Network and Distributed Systems Symposium*, 2016.

[16] N. Šrndić and P. Laskov, “Practical evasion of a learning-based classifier: A case study,” in *the IEEE Symposium on Security and Privacy*, 2014.

[17] M. Christodorescu, S. Jha, and C. Kruegel, “Mining specifications of malicious behavior,” in *Proceedings of the 1st India Software Engineering Conference*, ser. ISEC, 2008.

[18] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda, “Scalable, behavior-based malware clustering,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2009.

[19] E. Vasilomanolakis, S. Karuppayah, M. Muhlhauser, and M. Fischer, “Taxonomy and survey of collaborative intrusion detection,” *ACM Comput. Surv.*, 2015.

[20] D. Dash, B. Kveton, J. M. Agosta, E. Schooler, J. Chandrashekar, A. Bachrach, and A. Newman, “When gossip is good: Distributed probabilistic inference for detection of slow network intrusions,” in *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006.

[21] Y. Xie, H. Kim, D. R. O’Hallaron, M. K. Reiter, and H. Zhang, “Seurat: A pointillist approach to anomaly detection,” in *Recent Advances in Intrusion Detection*, 2004.

[22] “Advanced malware protection (amp),” http://www.cisco.com/c/en/us/products/security/advanced-malware-protection/index.html.

[23] “Advanced malware protection and detection (ampd),” https://www.secureworks.com/capabilities/managed-security/network-security/advanced-malware-protection.

[24] S. Fortunato, “Community detection in graphs,” *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[25] J. Xie, S. Kelley, and B. K. Szymanski, “Overlapping community detection in networks: The state-of-the-art and comparative study,” *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, p. 43, 2013.

[26] “Statement regarding cyber attack against anthem,” http://www.sophos.com/en-us/threat-center/mobile-security-threat-report.aspx.

[27] “Symantec report on black vine espionage group,” http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the-black-vine-cyberespionage-group.pdf.

[28] http://www.fraudwatchinternational.com/phishing-alerts.

[29] “Symantec intelligence report,” https://www.symantec.com/content/en/us/enterprise/other_resources/b-intelligence-report-01-2015-en-us.pdf.

[30] “Wasserstein metric,” https://en.wikipedia.org/wiki/Wasserstein_metric.

[31] S. Vallender, “Calculation of the wasserstein distance between probability distributions on the line,” *Theory of Probability & Its Applications*, vol. 18, no. 4, pp. 784–786, 1974.

[32] J. Benamou and Y. Brenier, “Mixed l2-wasserstein optimal mapping between prescribed density functions,” *Journal of Optimization Theory and Applications*, 2001.

[33] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff, “A sense of self for unix processes,” in *Security and Privacy, 1996. Proceedings., IEEE Symposium on*, 1996.

[34] D. Wagner and P. Soto, “Mimicry attacks on host-based intrusion detection systems,” in *the ACM Conference on Computer and Communications Security*, 2002.

[35] M. Bailey, J. Oberheide, J. Andersen, Z. Mao, F. Jahanian, and J. Nazario, “Automated classification and analysis of internet malware,” in *Recent Advances in Intrusion Detection*, 2007.

[36] W. Robertson, F. Maggi, C. Kruegel, and G. Vigna, “Effective anomaly detection with scarce training data,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2010.

[37] S. S. Clark, B. Ransford, A. Rahmati, S. Guineau, J. Sorber, W. Xu, and K. Fu, “WattsUpDoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices,” in *USENIX Workshop on Health Information Technologies*, 2013.

[38] G. Gu, J. Zhang, and W. Lee, “Botsniffer: Detecting botnet command and control channels in network traffic,” in *Presented at the 16th Annual Network & Distributed System Security Symposium*. NDSS, 2008.

[39] C. Kolbitsch, P. M. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, “Effective and efficient malware detection at the end host,” in *Proceedings of the 18th Conference on USENIX Security Symposium*, 2009.

[40] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, “Crowdroid: Behavior-based malware detection system for android,” in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, ser. SPSM, 2011.

[41] V. Vlachos, S. Androutsellis-Theotokis, and D. Spinellis, “Security applications of peer-to-peer networks,” *Comput. Netw.*, vol. 45, no. 2, 2004.

[42] C. Krüegel, T. Toth, and C. Kerer, “Decentralized event correlation for intrusion detection,” in *Proceedings of the 4th International Conference Seoul on Information Security and Cryptology*, ser. ICISC ’01. Springer-Verlag, 2002.

[43] M. Locasto, J. Parekh, A. Keromytis, and S. Stolfo, “Towards collaborative security and p2p intrusion detection,” in *Information Assurance Workshop, IAW*, 2005.

[44] V. Shmatikov and M.-H. Wang, “Security against probe-response attacks in collaborative intrusion detection,” in *the Workshop on Large Scale Attack Defense*, 2007.

[45] J. Bethencourt, J. Franklin, and M. Vernon, “Mapping internet sensors with probe response attacks,” in *Proceedings of the 14th USENIX Security Symposium*, 2005.

[46] Y. Shinoda, K. Ikai, and M. Itoh, “Vulnerabilities of passive internet threat monitors,” in *Proceedings of the 14th Conference on USENIX Security Symposium*, 2005.

[47] “Data binning,” https://en.wikipedia.org/wiki/Histogram.

[48] D. Kirat, G. Vigna, and C. Kruegel, “Barecloud: Bare-metal analysis-based evasive malware detection,” in *Proceedings of the 23rd USENIX Conference on Security Symposium*, 2014.

[49] “Kaspersky security bulletin 2015,” https://securelist.com/files/2015/12/Kaspersky-Security-Bulletin-2015_FINAL_EN.pdf.

[50] “G4 - yahoo! network flows data,” https://webscope.sandbox.yahoo.com.

[51] “Enron email dataset,” https://www.cs.cmu.edu/~./enron.

[52] F. Kooti, L. M. Aiello, M. Grbovic, K. Lerman, and A. Mantrach, “Evolution of conversations in the age of email overload,” in *Proceedings of 24th International World Wide Web Conferenced (WWW)*, 2015.

[53] S. Foss, D. Korshunov, and S. Zachary, “An introduction to heavy-tailed and subexponential distributions,” 2009, springer Series in Operations Research and Financial Engineering.