When Malware Changed Its Mind

An Empirical Study of Variable Program Behaviors in the Real World

Erin Avllazagaj, Ziyun Zhu[+], Leyla Bilge[*], Davide Balzarotti[†], Tudor Dumitraș
University of Maryland, College Park
[+]*Facebook*
[*]*NortonLifeLock Research Group*
[†]*EURECOM*

Abstract

Behavioral program analysis is widely used for understanding malware behavior, for creating rule-based detectors, and for clustering samples into malware families. However, this approach is ineffective when the behavior of individual samples changes across different executions, owing to environment sensitivity, evasive techniques or time variability. While the inability to observe the complete behavior of a program is a well-known limitation of dynamic analysis, the *prevalence* of this behavior variability in the wild, and the *behavior components that are most affected* by it, are still unknown. As the behavioral traces are typically collected by executing the samples in a controlled environment, the models created and tested using such traces do not account for the broad range of behaviors observed in the wild, and may result in a false sense of security.

In this paper we conduct the first quantitative analysis of behavioral variability in Windows malware, PUP and benign samples, using a novel dataset of 7.6M execution traces, recorded in 5.4M real hosts from 113 countries. We analyze program behaviors at multiple granularities, and we show how they change across hosts and across time. We then analyze the invariant parts of the malware behaviors, and we show how this affects the effectiveness of malware detection using a common class of behavioral rules. Our findings have actionable implications for malware clustering and detection, and they emphasize that program behavior in the wild depends on a subtle interplay of factors that may only be observed at scale, by monitoring malware on real hosts.

1 Introduction

The ability to understand and model *malware behavior* plays a key role in many security applications. This typically involves executing samples inside an instrumented environment, designed to collect system and API call traces that can be further analyzed to reconstruct the runtime behavior. Such behavioral analysis methods have been applied to detecting [10,12,16,23,24,32] new or polymorphic malware for which static analysis fails [37,48], and to clustering samples into malware families [5,7,40,41], in order to identify the malicious behaviors that characterize each family. However, the effectiveness of all these methods depends on their ability to identify invariant parts of the behavioral traces. In consequence, variations in the observed malware behavior, which may arise from adversarial intent [6,20] or biases in the data collection [43], can result in models that overfit the analysis environment and fail to generalize to the behavior observed in the wild. This problem, which is a consequence of the limitations of dynamic analysis, is widely accepted among researchers and practitioners as a fundamental challenge for behavioral analysis.

Unfortunately, *just how much the behavior of malware varies in the wild* is a largely open question, outside of a few prominent and well studied malware families. A common approach to accounting for behavior variability is to acquire multiple samples of the same family and to analyze their executions together, in order to extract the common behavior patterns of the malware family. However, if the *behavior of individual samples varies*, across *different hosts* and across *time*, the common patterns extracted will not be representative of the malware's behavior on real hosts. Additionally, the behavioral traces are typically collected by executing the malware in a controlled environment [1,2,17,52], in order to prevent it from harming other hosts. If the behavioral models are created and tested with traces collected in the same environment and during the same time period, artifacts that only manifest under those conditions will inflate the apparent effectiveness of those models and give a false sense of security.

It has been challenging to measure per-sample variability systematically, despite the fact that researchers and practitioners have known about it for over a decade. For example, Lindorfer *et al.* reported that one sample's behavior may change across execution environments because of different OS versions and libraries [31]. Other researchers studied the evasive techniques implemented by malware authors to ensure that traces collected in a sandbox environment are not representative of its behavior in the real world [6,20]. Rossow *et al.*

reported how downloader behaviors change over time, owing to time bombs or new instructions received from the command and control (C&C) channel [42]. These prior studies have confirmed the existence of per-sample behavior variability and showed its potential impact. However, because they were conducted in experimental infrastructures, they did not reveal the prevalence of this variability in the wild, or which components of the sample's behavior are most likely to vary. How much, and in what ways, the behavior of benign programs varies in the wild are also open questions. The prior research has also showed that the effectiveness of malware-detection models degrades over time, as new samples exhibit previously unseen behaviors [19, 38, 47]. Previously unseen behaviors of the samples *already covered* by the model may similarly degrade the detection performance, but this effect has not been quantified before.

In this paper we conduct the first study to understand and measure the variability in the behavior of malware and potentially unwanted programs (PUP) at scale. We focus on API- and system-call based behavioral profiles, and we conduct a quantitative analysis of per-sample behavioral differences on end hosts. To this end, we use a unique dataset of 7.6M execution traces, recorded in 5.4M real Windows hosts from 113 countries. At the time when the data was collected, it was not known whether the samples were benign or malicious. The samples were executed by the users, who interacted with them naturally, and the behavioral monitoring and analysis was employed as a last line of defense against unwanted behaviors.

We measure the variability in the behavior of samples later determined to be malware and PUPs, and we compare it to a baseline we draw from the benign samples. Across executions recorded on different hosts we found that the number of *actions* performed (e.g. the creation of a new file or the modification of a registry key) varies 6$\times$ more for malware than for benign samples, and this difference increases to 15$\times$ when looking at the number of created files. In contrast, different executions recorded weeks apart on the same host do not show such a high range of action variability. When considering action *parameters*, (e.g. file names), we observe little to no variability across time for benign samples (the action parameters tend to remain constant on the same machine), and a very large variability for malicious samples (the intersection of the common values is almost empty).

We further assess the challenges for identifying the *invariant* parts of per-sample behaviors, which have implications for building behavioral rule-based detection signatures, and for clustering samples into malware families. We show that, when building rules that use actions and tokenized parameters, the information collected from a single execution is inconclusive, but it is possible to observe most of the behaviors from a few traces. For instance, file names extracted from three different hosts cover, on average, 90% of the executions and using more than four traces provides diminishing returns. We also show that, when performing a malware clustering experiment, one third of the samples exhibit sufficient variability in behavior that their traces appear in multiple clusters. As this would not be observed when using a single trace per sample, our result suggests that the accuracy of mapping samples to the correct family, through clustering, is lower than previously believed.

These findings emphasizes that real malware behavior depends on a subtle interplay of factors, such as environments, time, and user interactions, which cannot be observed by executing the sample once in a sandbox environment. We discuss the actionable implications of our results and the alternatives to account for behavioral variability. More importantly, these results emphasize the unique insights that we can gain by monitoring malware behavior at scale, on real hosts. Importantly, such monitoring can be performed ethically by anti-virus systems. This radical shift from the way behavioral analysis is conducted today may bring a degree of external validity that sandboxes cannot provide.

In summary, we make three contributions:

- We analyze program behavior at scale, using 7.6M call traces recorded in 5.4M real hosts. These traces include natural user interactions with the programs and have high external validity compared to the prior work.
- We study how the behavior of individual samples changes across hosts and time, and we compare the variability of Windows malware, PUP, and benign programs at multiple granularities.
- We analyze the invariant parts of the malware behaviors, and we show how this impacts a common class of behavioral rules for malware detection.

2 Problem and Methodology

The main goal of malware analysis is to identify and characterize the behavior of unknown samples such that behavioral indicators that are specific to a malware family could be used for malware detection or classification. Because the behavior of executables could vary depending on when, where and at what setting it is executed, part of the behavior for any given program is transient in nature.

In our dataset, we observed that some executions of the Ramnit worm [39], result in the creation of a large number of mutexes. The reason is that the worm uses a privilege escalation exploit, which creates a lot of mutexes, only if it is executed in user-mode on a vulnerable version of Windows 7. If instead Ramnit is executed with admin privileges or within a different Windows version, the malware would not perform the exploit. If an analyst, or an automated system, created a signature by looking at the behavior collected on Windows 7 (a popular choice by many malware analysis sandboxes), those mutex creations could be used for constructing the signature. However, these actions would only appear in a fraction of end user machines, thus resulting in a poor detection coverage.

To mitigate this problem and identify truly invariant parts of malware behavior, it is important to collect malware executions across multiple machines, as suggested by Rossow et al. [43] and over time, as suggested by Pendlebury et al. [38]. However, prior works does not make concrete recommendations for the most optimal set up (e.g, the optimal re-execution interval, the number of different machines, the number of different OSes, etc.) that allows those invariant parts to be identified accurately. Our goal is to fill this gap in the state-of-the-art.

Despite these very time-consuming therefore costly suggestions, the industry practitioners often choose to aggregate behavior of different samples of a family for signature generation [11, 24]. However, the majority of malicious samples cannot be mapped to a known family (malware with generic labels are 1.3 times more common than those that belong to a well-defined family) [27], making it impossible to perform such an aggregation.

To shed light on the magnitude of this problem, we analyze $7.6M$ executions out of which $3.1M$ belong to malicious and unwanted programs and the rest to benign. In total, the executions of each sample span at least 10 machines, while 45% appear at least 1 week from the sample's first appearance. This measurement, the first of its kind, allows us to assess the amount of behavior variability in the wild, and to study the minimum number of experiments required to rule out transient behaviors and derive signatures that achieve the highest coverage on end hosts, filling a crucial gap in the state-of-knowledge about the most optimal execution configurations for signature generation.

2.1 Measuring Variability

We describe the behavior of a sample through its interactions with the host Operating System. Because a semantic interaction, such creating a new file or spawning an OS process, may be accomplished with various system or API calls, and the calls differ across OS versions, we abstract these interactions as *actions*. Our actions model high-level operations, such as process injection, file creation, or the modification of a registry key; we report all the action types analyzed in Section 3. An action may have one or several *parameters* to specify the target that the action is operating on (e.g. the registry key being modified), as well as the actual value it writes or modifies (e.g. the value written in the registry). An *execution trace* for a sample consists of a sequence of actions and the corresponding parameters. The traces captured by malware detectors based on both system calls [10, 34] and native API calls [6, 7, 11, 18, 21] can be mapped to action-based execution traces.

We measure variability at two levels of granularity. First, we count the actions in an execution trace and compute the *action variability*. We maintain separate counts for each action type, as well as for all the actions taken together. We then compare these counts across all the execution traces of a sample, using several measures of variability as described below. This provides a conservative assessment of variability, indicating for example when a sample creates one file on a host and two on another. We report *how much* action variability we observe, *which action types* account for most of the variability, and how these the variability changes across *space* (a sample executing on different hosts in the same week) and *time* (a sample executing on the same host in different weeks). We also compare the action variability in malware, PUP, and benign samples.

Second, we compare the action parameters coming from different execution traces of the same sample, using measures of set similarity. This *parameter variability* allows us to identify differences among executions when the number of actions remain the same, for instance when a sample creates a file with different names on each host. This comparison provides further insight into the semantics of the variable actions; for instance, we identify which parameter parts (e.g., the filename vs the directory path) differ among different executions.

Measuring action variability. The action counts coming from different execution traces of a sample form an empirical distribution. We can characterize this distribution using various measures of location (e.g. mean, median, mode) and spread (e.g. variance, standard deviation, median absolute deviation, interquartile range); we are interested in the latter when assessing action variability. The main challenge in selecting a statistical measure of spread is to avoid drawing incorrect conclusions because of outliers in the distribution.

We illustrate this challenge by showing how different variability measures perform over the executions of one sample of AutoPico, a Windows piracy software. Usually the sample creates four files when executed: two `log` files, one `dll` and one `.sys` file. However, in six traces out of 62, AutoPico only dropped the two log files (because the samples was unable to execute correctly), and in four traces it created more than 15 times the same log files in the same location (possibly due to the fact that the sample modified more registry keys, each time recreating the log file from scratch). The ordered list of the number of file creation events for all executions in our dataset looks like the following:

$$[2,2,2,2,2,2,4,4,4,\ldots,4,17,19,19,20]$$

The Interquartile Range (IQR) and the median absolute deviation (MAD) are measures of spread that are robust to outliers. Unlike the classic standard deviation, these measures are not affected by measurement values that are either too low or too high. For this reason, IQR and MAD are widely used in other experimental fields [26, 30]. In our study, a trace may exhibit an atypical number of actions owing to the malfunctioning of the sample, because of the lack of a required component or because the host was shut down mid-execution. High action counts may also occur when the malware was designed to infect all of the files in a directory and it encounters a few machines with an unusually large number of files,[1] which results in outliers for the number of file actions. The IQR is the difference between the 75^{th} and 25^{th} percentile values of the action-count distribution. In the AutoPico example, the IQR is 0, as it is the difference of the value in the 47^{th} position (a 4 in our example) and that in the 16^{th} position (again a 4) in the ordered list of 62 values. The MAD is the median of the absolute values of each count's deviation from the median. In the example the MAD is 0 as well, because, after subtracting 4 (the median) from each count, we get a vector where 0 is repeated 52 times and there are only 10 non-zero values, and the median of this vector is 0. In contrast, the standard deviation for the AutoPico traces is 3.67, which inflates the action variability that would be reported. Moreover, the variability would be heavily influenced by the four large outliers (17, 19, 19, 20): without them, the standard deviation of the action counts would drop 0.61, while the IQR and MAD would remain 0. This suggests that robust measures of spread, such as the IQR and MAD, are not likely to lead to conclusions biased by artifacts in the data.

At the same time, the tail of the distribution may also provide meaningful insights, e.g. when it reflects the behavior of targeted malware. We therefore select two additional measures, the 90-10 and 99-1 percentile ranges, because they are analogous to the IQR but are gradually less conservative in discarding the distribution tails. In the AutoPico example, the 90-10 and 99-1 percentile ranges are 0 and 17 (19-2) respectively. In our analysis, we compute the MAD, and the 75–25 (IQR), 90–10 and 99– 1 percentile ranges. We report one representative measure when the results are similar, and we discuss when we observe differences among the four measures.

Measuring parameter variability. We measure variability on parameters for each action type separately. We then compute the Jaccard index, which is a popular choice to measure the distance between the parameters observed in two malware executions [22, 31], on the parameters observed in different executions of each sample. This way it is possible to identify whether similar parameters are chosen (e.g., create a file with the same filename) or on contrary, the parameters are randomized or very different among executions, therefore, malware detection signatures should not incorporate them. We also perform IQR measurements on the count of unique parameter values,to get a precise picture of what, and how, changes across multiple executions.

[1] An example is the authors' fileserver, which stored so many files at one time that it crashed our backup service.

```
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        CommandLine: '*-noni -ep bypass $*'
    condition: selection
```

Figure 1: A sample signature from SIGMA.

2.2 Finding Behavioral Invariants

As we introduced in the previous section, actions are a common abstraction to represent units of behavior. On top of that, researchers have proposed many different models to build signatures by expressing patterns over sets of actions. While the literature of models is very rich, ranging from simple ngrams or ordered bags [10] to complex graph-based structures [24], the industry still lacks a common framework for expressing and sharing behavioral models (the role that Yara [3] plays for static signatures).

To the best of our knowledge, the only available resource that contains a sufficiently-large set of signatures of this kind is provided by SIGMA [44], a project that proposes a language to express patterns for log analysis. As OS audit logs contain information about the interaction of each process with the environment (something equivalent in nature to system calls or the abstract actions in our model) the language used to express SIGMA rules allows analysts to write Yara-like pattern over the runtime events of a sample.

By reviewing previous papers and the SIGMA ruleset, we found that a common building block of all these signatures is the ability to check for the presence of an action and match a portion of the its parameters (typically through a regular expression). For instance, Canali et al. [10] use the action type and the full parameter value to create complex signatures. Similarly Trinius et al. construct a representation of malware behavior that uses the action type and parts of the parameters to create a behavior profile for their malware and Trinius et al. [51] use an exact match on their proposed features. This is also the case for SIGMA rules, as the one reported in Figure 1, which matches a process creation action in which the command line parameter matches the specified pattern.

Our goal is not to create signatures nor to evaluate different signature models. Instead, we want to measure which constant elements exist across multiple executions, with the assumption that any good signature would need to build upon these elements and avoid using information from other transient behaviors.

For this reason, we break down each parameter value in a set of tokens according to classic windows delimiters [49]—such as backslashes for directories and spaces for command-line arguments—and study the evolution of each token both

	Mal	PUP	Ben
Num. samples	2424	1621	22443
Num. machines	0.5M	0.9M	4M
Num. executions	1.1M	2M	4.5M

Table 1: Dataset summary.

individually and aggregated with other tokens extracted from the execution traces. As an example, we observed that around 70% of the SIGMA rules contain at least one of such token, confirming their role of building blocks for more complex signatures.

3 Dataset

The dataset we are using is a collection of 7.6M execution traces that the AV vendor has collected across 5.4M real users during the year of 2018. The data is collected by a component of the AV engine that is responsible for behavior-based detection. This component records high-level behavioral data about the executed programs until they terminate or until the system is able to classify them as either benign or malicious and kill. For the sake of validity, our data only includes programs that terminate normally. Therefore, unlike data collected from sandboxes, our data is not limited to few minutes of execution and because the traces are collected from real users, they do not suffer from the limitations introduced by synthetic analysis environment. Our data does not consist of malware samples that were executed intentionally for data collection, but samples that at the time of collection were not yet known to be malicious or pup and were able to evade the static malware detection solution installed on the computers. Our data is a reflection of the set of threats with which the behavioral detection components need to combat.

Dataset coverage statistics. Each item in our data consists of a sequence of behavioral actions performed by a sample together with SHA-256 hash of the sample, an anonymous machine identifier and a timestamp. Thanks to the unique SHA-256 hashes of the samples, we were able to query VirusTotal (VT) in the following year (2019) of the data collection and identify the corresponding labels assigned to those samples by various AV engines. While we labeled the samples that were consistently labeled as benign by all AV products, we label samples as malicious or PUP using AVClass [46], a state-of-the-art technique for massive malware labeling. From the VT reports obtained in August 2019, AVClass identified $22,443$ benign, $2,424$ malware and $1,621$ PUP samples, as listed in Table 1. We perform our variability analysis on execution flows we were able to label with high confidence and those that were observed in at least 10 machines. This experimentally chosen threshold made it possible to accurately

	Ratio
Windows 7	56%
Windows 10	35%
Windows 8.1	3.1%
Windows Server	2.6%
Windows XP	2%
Other Windows Versions	1.3%

Table 2: OS version distribution.

measure variability of the sample sample across different machines. 85% of the samples were executed between 10 and 100 machines, rest were observed in more than 100. The data was collected from computers from across 133 countries: USA(48%) and China (14%) have the largest fraction of our data points.

In Table 2 we show the distribution of the Operating Systems for the machines in our dataset. The vast majority of machines run the Windows 7 build 7601 and the rest run a flavor of Windows 8.1 or 10. 55% of the executions happen less than one week apart, while respectively 12%, 6%, 4% and 3% are executions that were collected after the second, third, fourth and fifth week from the initial recorded execution. On the 11% of the samples' re-execution happens 9 weeks after the first appearance of the malware. As a matter of fact, we measure the time variability for the executions happening during the first 4 weeks after the first appearance, covering over 80% of the executions. For instance, a crypto miner sample first appeared on April 5th and within 7 days we had 47 executions from 35 machines. During the next 7 days we captured 18 executions, 4 of which on new machines. In the 3rd week we record the last 7 executions, none of which from any new machines.

Execution statistics. An execution trace is composed of multiple actions. The actions are heuristically-defined behavior units such as file creation/modification, registry key creation/modification, mutex creation etc. In addition to those common behaviors analyzed largely by the literature, we also have some behavioral actions that were defined by the security vendor such as disable Windows defender, disable updates, change firewall options, keylogging, change IE settings etc.

In table 3 we show the top 8 action types in our dataset which corresponds to 87% of the whole data. On average, per execution trace we identified 150 actions out of which 39 are file creations. In our study, due to space constraints we present the action level variability analysis for a subset of these actions. To this end, we set the following criteria:

1. **Action occurs in any execution in more than 25% of the machines.** To measure a non-zero machine variability of a certain action with IQR, it is necessary to observe it at least 25% of the machine.

	Ratio of dataset
File Create	26%
Mutex Create	20%
Registry Set	14%
ProcessLoad	8%
PECreation	6%
RegistryKeyCreated	6%
DirectoryCreated	4%
ServiceCreation	2%
Others	14%

Table 3: Ratio of action types over all the dataset.

2. **More than 1 action appears in the executions.** If the action happens only once in executions, it is not possible to measure its variability (the only possible result could be 0 or 1.

7 of the actions in Table 3 meets this criteria. More details for other actions in our dataset are provided in Tables 8 and 7 in Appendix.

Ethical Considerations. As mentioned earlier, we did not distribute or launch any samples on the user machines; instead, all the executions in our data set were triggered by the users, and the malware and PUPs we report reflect real-world attacks agains those machines. The anti-virus detects and blocks all the malicious samples known at the time and collects execution traces for samples that remain suspicious, in a last-resort effort to discover unknown malware. In consequence, we do not cause any harm to the machines in our study that would not have occurred without the anti-virus product and our data collection. Moreover, future updates to the anti-virus will clean the infected hosts once the malware is discovered, owing to the data collection. The behavioral analysis data was collected from users who opted in sharing their data. Necessary anonymization actions are taken to preserve the privacy of the customers. None of the data fields in the dataset contains any identifiable information.

4 Behavior Variability in the Wild

In this section, we analyze the variability we observed in the behavior of malware, PUP, and benign programs when executed on different end-user machines. We measure both the action variability and the parameter variability, as discussed in Section 2.1. We first conduct these measurements across space (the differences among a sample's execution traces on different machines) and time (the differences among its traces in different weeks).

In our data, some executions contain duplicates (i.e., the same action type with the same parameter value). This could happen for example when a sample opens the same file multiple times. As these operations are idempotent with respect to the our behavioral specifications, defined in Section 2.1, we perform deduplication on our data before we apply the variability analysis.

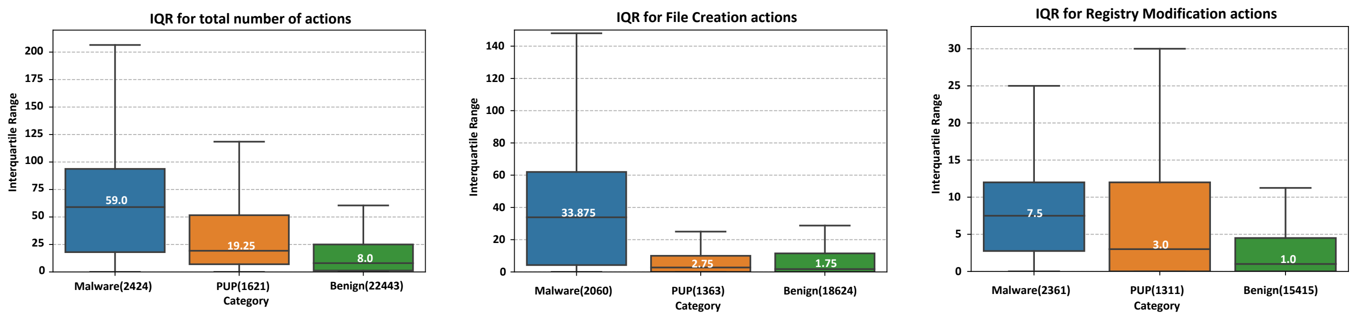
4.1 Machine Variability

We start by measuring the variability of executions of the same sample across different machines. Our goal here is to understand whether this phenomenon exists and if it does, on which type of executables it is more prevalent. We only look at executions of a sample that happen max one week apart to identify variability that happen only due to being run on different machines not due to time.

4.1.1 Action Variability

We analyze action variability through IQR and MAD. In order to understand the impact of the outliers on the results, we also look at the difference among 90-10 and 99-1 percentiles. Figure 2 illustrates the distribution of IQR variability, across all actions (Figure 2a) and only for the two most common actions we observed in our dataset: file creation (Figure 2b) and registry modification (Figure 2c). The numbers between parentheses for each category is the number of samples that we use for our variability analysis. Because not all samples had file creation or registry key modification actions in their executions, these numbers are lower than the total number of samples (Table 6 in the appendix provides a detailed breakdown of action variability). The separate boxplots for malware, PUP and benign samples allow us to compare the action-variability distributions within these categories and to assess the extent of these differences. To confirm these visual observations we compare these empirical distributions using pairwise U-tests [33], a non-parametric method for inferring whether the samples are likely drawn from distinct distributions. In the paper, we report differences that are statistically significant at $p < 0.001$ level.

Malware exhibits higher variability across machines. We expect to see a higher behavior variability in malware samples, owing to a host targeting, evasion and obfuscation attempts, or the tendency to attempt operations that may fail on some hosts (e.g. privilege escalation). Figure 2a confirms this: comparing the three boxes, which represent the bulk of the measurements from each empirical distribution, suggests that the action-variability of malware is typically higher than that of PUPs, which is typically higher than that of benign programs. The median IQR for malware is 59 actions, which means that the top 25% of a sample's execution traces are > 59 actions longer that its bottom 25% traces, for half of the malware samples in our dataset. In contrast, the median IQRs are 19.25 and 8 actions for PUP and benign, respectively. We observe similar trends with the MAD measurements. While



(a) (b) (c)

Figure 2: IQR action variability for all actions and the two most common actions in our dataset.

the average difference from the median for malware is 35.5 actions, for PUP and benign, it is 10.3 and 2.9 respectively.

This leads to the question *where does this variability come from*? When breaking down the variability according to action types, we observe a striking difference for file creation actions (Figure 2b). The median IQR for malware is 15$\times$ larger than for PUP and benign samples, and the bulk of the distribution includes much larger values. In contrast, the variability distributions for PUP and benign samples do not appear to be different for file-creation actions, while PUPs exhibit more variability for registry-modification actions (Figure 2c). This suggests that malware classification solutions based on file creation actions could lead to inaccurate results, as the high variability among the execution traces of a sample may place some of these traces in different clusters; we investigate this in more depth in Section 6. Conversely, a malware detector able to observe executions on multiple hosts could utilize file-creation variability as an indicator of malicious behavior.

Case study. We refer back to the Ramnit sample in Section 2 At least 25% of the executions occur on Windows 7 machines where the malware is running with user privileges. Therefore, the malware runs a privilege escalation exploit causing a large number of mutex creations. The rest of the executions happen in different OS version or with admin privileges, thus the executions are shorter. The action variability is affected by the longer executions showing an IQR of 34, which is the number of mutex creations.

There is a significant variability on the behavior of malware among different machines. When malware detection solutions rely on data collected only from one sample up to 200 (med. 59) behaviors could be underrepresented in the detection model.

Even though malware still shows a significantly higher variability when expressing the variability in terms of the 90–10 and 99–1 percentile ranges instead of the IQR, the action-variability distribution of malware becomes harder to distinguish from the PUP and benign distributions This is not surprising as these measures are not as robust to outliers as the IQR and MAD. The PUP and malware distributions remain distinguishable for file-creation actions, but overlap significantly for registry-modification and mutex-creation actions. In fact, a few PUP samples seem to have more extreme outliers causing the 99–1 range to be larger than that of malware.

		Median			**75th percentile**		
		Mal	**PUP**	**Ben**	**Mal**	**PUP**	**Ben**
File	Path	4	1	-	10	3	2
	Name	25	2	1	49	8	8
	Ext.	3	1	-	5	2	1
PE	Path	1	-	-	1	1	-
	Name	1	-	-	3	2	1
	Ext.	1	-	-	1	1	-
RM	Key Path	2	1	-	3	3	1
	Key Name	5	2	-	9	6	2
	Value	5	2	1	10	6	3
D	Path	1	-	-	1	1	-
RC	Path	2	1	-	3	3	1
MC	Name	6	3	1	9	7	4
P	CMD line	4	-	-	6	1	1

Table 4: Parameter(s) IQR variability for malware, PUP and benign.
[**PE**] PE File Creation actions, [**D**] Directory Creation, [**RM**] Registry Key Modification, [**RC**] Registry Key Creation, [**M**] Mutex Creation, [**P**] Process Creation actions.

While malware and PUP both vary more than benign in all the action types, they show variability in different action types.

4.1.2 Parameter Variability

We now take a closer look at the variability in the parameter values. We calculate the Jaccard index among the parameters of the same actions types in the execution traces. In this experiment use the full value of the parameters types listed in Table 4. Our observation was that the parameters values across different machines have a high variability for both malware, PUP and benign programs. For all of the parameter types, we obtain a Jaccard index is 0, indicating that there is no full value shared across all executions. Note that for this step, we do not normalize the data to remove computer specific artifacts and also do not extract substrings from the parameters. However in Section 5, when exploring invariants parts among executions of samples, we will perform deeper investigation on the substrings as well.

No parameter value is shared among all machines except for file extension. An analyst has to rely on substrings/tokens of the parameter values to create signatures.

We also perform IQR measurements, to obtain more insights about the distribution of the variability among different parameter types. In Table 4 we report the median and 75th percentile IQRs of different parameter types. We can clearly see that benign programs rarely change the directory they work on, the file names created or the executables they launch. On the other hand, malicious samples tend to create a significant amount of new files (25 new files for 50% of the malware), to work on several directories and even to create different executables over different executions. This finding indicates that the same sample's behavior can vary to an extent that it could be hard to identify a behavioral indicator that is common among all. We will explore this aspect in more detail in the following section.

Case study. In the executions of a Glupteba malware sample we observed that the Jaccard index across multiple machines is 0 for file names and 0.2 for mutex names, while the IQR for file and mutex creation is 0 and 2 respectively. This means that the malware changes significantly the name of files but not their absolute number, while mutex names are more similar but with a larger variability in terms of number. In this particular case, mutexes were a better candidate for building signatures, as we found that `h48yorbq6rm87zot` appeared in all the machines, which is also confirmed by a report from TrendMicro [50]. On the contrary, the mutex `ZonesCacheCounterMutex` and `ZoneAttributeCacheCounterMutex` only appeared in half of the machines, which explains the IQR of 2.

4.2 Time Variability

In this section we look at how variability is impacted by the time in which a sample is executed. We start again by looking at the volume of actions and then zoom into those actions by including their parameters into the analysis.

4.2.1 Total Action Variability

We measure the action variability by comparing executions of the samples in different weeks. Since 80% of the samples in our data were executed at most four weeks after the first week of their appearance, we perform the per-week time analysis on those next 4 weeks. To simulate what an analyst would deal with, we consider the first week's executions as the base and compare it with each of the 4 consecutive weeks. Here, we cannot use IQR as for each sample we only have 4 data points. We simply count the number of missing and new actions observed in each sample's executions compared to the previous week. The results are reported in Figure 3.

Malware have the highest number of missing and additional actions. The general takeaway for coarse-grained time variability analysis is that there is a significantly larger time variability in malware compared to PUP and benign samples.

According to the Figure 3, on average across all machines we see 6 missing actions 1 week after the first execution. Even though this number might seem low, depending on what those actions types are variability over time might have an impact on the malware detection solutions. Note that there are also some malware samples that show a tremendous variability (average of max being 17, max of max being 219). One possible explanation for this significant number of missing actions is that when malware is re-executed on the same machines, might not need to repeat some of the behavior such as creating particular files. At time of the data collection, the malware samples were not yet known and therefore, the machines were not cleaned up before the re-execution. However, we also observe variability over time when looking on the new actions that appear on the following weeks. Similarly, malware samples have on average 1 new action appearing every week, which is larger than PUP and benign. We also highlight that the machine with the maximum number of additional actions seems to have a maximum of 63 new actions and more than 3 new actions for 50% of the malware samples. For malware execution longer than 1 week from their first appearance less new actions appear, indicating a more stable behavior by time.

Case studies. A TOR-connected coinminer was dropping *miners.ini*, *miners.ini.** and *minergate.log* and launching *minergate-cli.exe* before January 18th in 2018. In some machines it was also dropping up to 14 **.tmp* files. After 18th this behavior completely stopped, resulting a number of missing actions in the following weeks. On the other side of the scale, we also identified a Remote Administration Tool, which initially dropped 5 dlls files and an executable (*setacl.exe*) before March 16th 2018. On April 3rd, it started dropping 7 more dll files and 3 new executables. We also have examples for malware that misses and adds new actions at the same. In it's first week of appearance the software was dropping various files on different machines such as *microsoft office*, *foxit pdf editor*, *autocad 2015 qqlivedownloader.exe*. This behavior could be due to the user's interaction with the malware or simply the malware hiding its purpose. The next week's executions no longer drop any of these files, but *zny_znykb030.exe* or *kuaizip_setup_2523474329_rytx2_001.exe* appears to download consistently in almost all the machines. We believe that

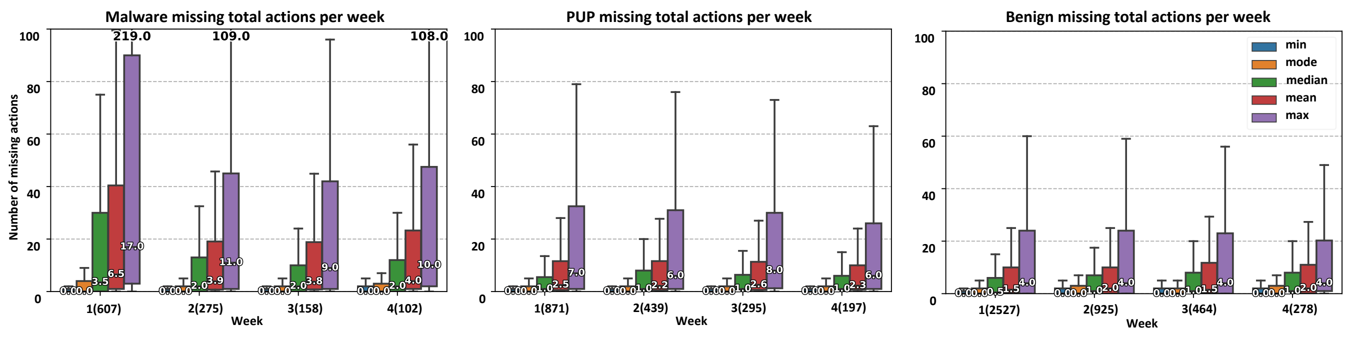


Figure 3: Missing actions compared to the previous week.

in this case the user had no control of the malware and the downloads happened silently. This malware was still running 4 weeks later performing the same actions. A final example is a sample of *Adware.Chinad*, which dropped various files (*microsoft office*, *foxit pdf editor*, *autocad 2015*). On the following week, a new executable (*zny_znykb030.exe*) followed by potentially pirated other software are downloaded.

In malware, time variability is the largest. While the variability is mainly due to the missing actions, there are also new events that appear on the following weeks.

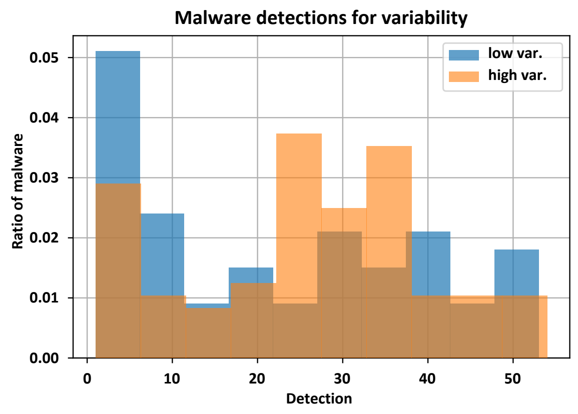


Figure 4: **Relation between detection and the time variability.** This result shows that malware that have high time variability have higher VT detections.

Malware with highest detection rates vary more over time. One interesting observation on the malware that exhibit more variability over time was that in general more of the AV engines would label them as malicious. In Figure 4 we show the distributions of the malicious samples with low time variability (lower than 25th percentile) against the ones with high variability (on the top of the 75th percentile). For each of samples, we check the total number of detections in VirusTotal in November 2019 (i.e., one year after the first time we observed the samples). As it can be seen seen, the samples that are AV software eventually detected the most had a higher variability across time. This shows that even if in general time variability is low across the board, for easier to classify samples it seems like time has a more significant effect on the variability.

To get a better understanding of this phenomenon we conducted two case-studies. In the 75th percentile we found a version of kuaizip and analyzed its behavioral data manually. This malware seemed to stop working sometime around the second week of April 2018, after which it still performed host-related actions but failed to download the PE files it was retrieving before. At the other side of the spectrum, we chose a malicious sample that exhibits low variability. We found an open source DLL injection tool classified as malware which performs exactly the same actions every time it runs. This likely-to-be-malicious sample injects into *roblox-playerbeta.exe*, creates *settings.xml*, and sets some registry keys. Upon further analysis we found that this is being intentionally used for cheating in games and this exact behavior is observed over and over again. While preliminary, these experiments seem to confirm that time variability affects the most those samples that rely on an external infrastructure.

4.2.2 Parameter Variability

When switching to the fine-grained analysis of each parameter, we now observe a very different picture from the results we obtained by looking at the variability among hosts. In fact, the Jaccard Indexes show that for goodware and PUP there are a large number of perfect matches over time when the sample is re-executed. For the full results we refer the reader to Table 5 in the Appendix.

Over time, malware actions parameters vary a lot, while PUP' and benign's ones do not vary at all. The difference is remarkable. Even at the median, the parameters of actions performed by benign software change very little, and the 75th percentile they change almost nothing at all. If we consider that the same indexes were zero when considering executions across different hosts, this result emphasize a very important distinction. Goodware creates different files, mutexes and registry keys in different machines. But when we consider two executions on the same machine, those values remain constant. The same phenomenon does not happens for malware, where the Jaccard index is zero across the board, both in case of different machines and in case of different executions on the same host. The only few exceptions to this rule are regarding file paths and file extensions, which still have a low similarity.

If we look at the 75th percentile things get more stable and both malware and benign files show a high similarity. This means that for at least 25% of the samples in our dataset we observe a stable set of parameters at different points in time.

At least 50% of the malware do not reuse same file names, registry keys values and paths, directories in their reexecutions, and 25% execute at least 1 new command.

5 Invariant Analysis

Our variability analysis confirms that malware behavior changes over time and on different machines. This indicates that if a behavioral malware detection system is designed with data collected at a fixed time or from a single computer with a particular configuration setting, the real behavior that is common to all possible executions might not be identified correctly. However, as we showed in Section 4.2.1, the fact that malware samples carry large variability across different executions does not rule out the possibility of building accurate detection models from behavior that remains stable. Therefore, in this section we focus on measuring the invariant part of malware behavior, to better understand how effective behavioral-based detection systems can be if their models are built upon the right set of events.

Roughly 80% of the SIGMA rules are created from values extracted from file and process creation events, and these two are also in top 7 most popular actions in our dataset. Therefore, due to space limitations, in this section we focus on those actions and their parameters. We identify the invariant behaviors only from the malicious samples as our goal is to evaluate behavioral malware detection techniques. We only use benign samples when simulating a signature generation process, by extracting the invariant parts that are not observed in the benign execution traces.

Beyond full parameter value. In Sections 4.1.2 and 4.2.2 we utilize the full value of the parameters to measure the jaccard index. In this section we follow a simple approach to split those values into smaller tokens, explained in Section 2.2, with the aim of finding a shared value across machines.

5.1 How Many Hosts Are Enough?

One of the main consequences of the findings discussed in Section 4 is that for building more effective and accurate signatures it is necessary to collect multiple data points rather

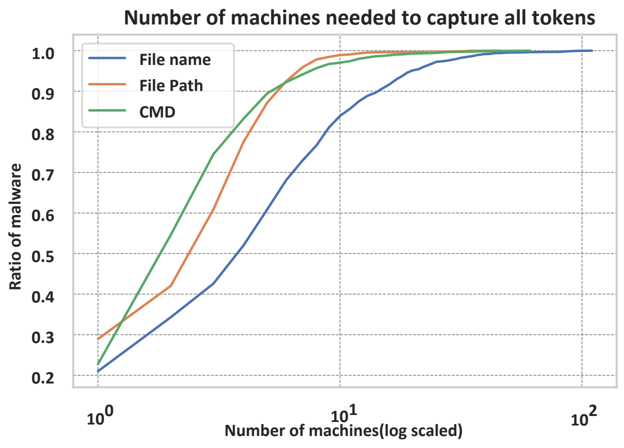


Figure 5: **CDF of number of machines and the amount of malware values.** It takes more machines to capture all the CMD tokens than other parameters' tokens.

than looking at a single trace collected from one environment. While this sounds intuitive, to our knowledge, there is no existing study that attempted to measure how many executions from different machines are needed to identify tokens that maximize the coverage of the generated signatures. To this end, we measure the number of executions of the same malware in the wild that can be detected by using the tokens extracted from a small set of executions, as well as the number of those executions that are needed to obtain all the malicious tokens. Based on that, we estimate how many machines are needed to achieve a high coverage of observed behaviors.

Figure 5 shows the empirical cumulative density function (CDF) of the fraction of malware samples for which we capture all tokens, for an increasing number of machines (x-axis). We only consider tokens that never appear on benign traces and we also exclude the unique tokens, i.e., those that only appear in one machine in the wild (as they could be the result of random values). Finally, we construct the CDF by adding first the traces that contain the highest number of tokens, and thus represent a conservative estimate. For 20% of the malware, we need only 1 machine to capture all the malicious command line tokens. If we increase the threshold to 10, we can identify all the command line tokens for 85% of the malware and all the file path tokens for 98% of the malware.

Naturally, the more machines we use the more tokens we can extract, but adding more machines provides diminishing returns. As we can see, for 21% of the malware we can capture all filenames in a single execution, and for 29% of them one execution is sufficient to discover all file path tokens. However, we need to collect 39 traces to observe all the malicious filename tokens that appear in 90% of the malware, while this decreases to only 11 and 17 machines for capturing file path and command line tokens respectively. Since the file path seems to converge faster, in Figure 6 we check the

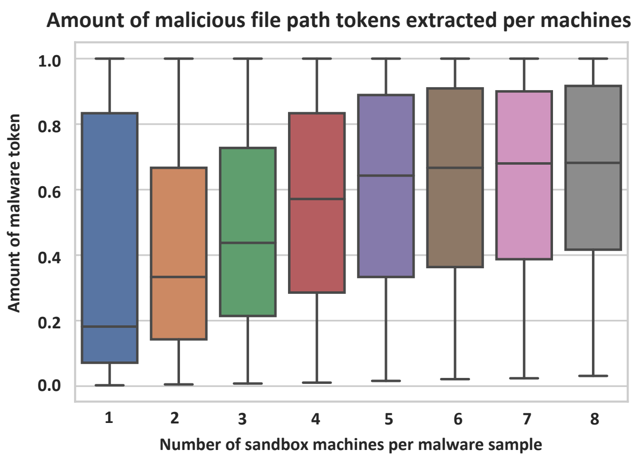


Figure 6: **CDF of number of machines and the amount of malware values.** After 4-5 machines we start to get diminishing returns in the number of new malware file path tokens discovered.

impact of the first 8 machines in the amount of tokens we are extracting. We observe that after 7 machines the return of investment becomes small, as for the top 50% of the malware samples we already extracted 68% of the tokens. When also check the other parameters and we noticed similar correlation between the amount of tokens extracted and the detection rate in Figure 7, which makes sense since having all the malware tokens means we have 100% detection.

While counting the new tokens can give an idea of how many traces we need to compensate for the diversity of behaviors, it does not tell us whether those tokens are sufficient or not to detect malware in the wild. Therefore, we conducted a second experiment. Here we use the tokens extracted from one execution to match the malware traces collected in other machines. If the combination of the tokens can cover all the other executions of the same sample, then we conclude that one execution is sufficient (in theory) to extract a perfect signature. If instead the extracted tokens cannot achieve 100% coverage, we add a second trace collected on a different (randomly chosen) machine in the same week (as for the moment we want to study the machine impacts and not the time impact) and we re-iterate the process. Since the result is dependent on the select machines, we repeat the experiments ten times and report the average.

From the boxplot in Figure 7a we can see that while for some malware one execution might be enough, in average the filenames extracted from one trace cover 82% of the executions and the value decreases to 77% if we use path information. However, the execution traces collected on three different machines are sufficient to achieve the highest coverage when using file name as the parameter. Similarly we find that it takes four machines to saturate the coverage for the command line and seven for the file path. The respective results can be found in Figures 7b and 7c.

Our results suggest that an analyst should analyze the malware in 3 random virtual machines to capture most of the file names, 4 for CMD line and 7 for file path. A possible way to generate such random machines, instead using the same machines for all malware, may be to use a random vm generator like SecGen [45] with the features proposed by Miramirkhani et al. [35].

5.2 How Soon Should We Re-Execute?

We now investigate the re-execution interval needed to achieve the best coverage in the wild. This is more difficult to measure, as it represents a trade-off. If you re-execute the sample too early, you may learn little and your signature may not catch the behavior that the malware will exhibit in the future. But if you re-execute the sample too far in the future, than your initial model might get outdated before you re-analyze the sample.

For this analysis, we take a first execution trace during the first week of appearance of the malware. Then we collect a second trace on the same machine, varying the time between one and four weeks in the future. We then use the tokens extracted from the first execution to match all malware executions until we re-execute the sample. From that time on, we incorporate the information of the second execution and update the signature to be used for future executions.

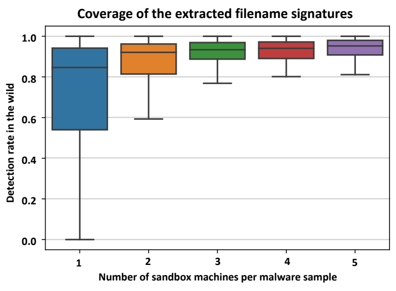
Figure 8 shows the results for the filename tokens. While the median detection does not change much, re-executing after three weeks provide more benefits (the minimum detection and the 25 percentile are much higher, which suggests that for some malware this makes a big difference).

For the file path the difference in the re-execution interval is smaller, which means that we need more machines to get better detection. However, even in this case we still notice a slightly smaller range when re-executing on the 4^{th} week, which means some malware show a different behavior around that time. The results are the same for command line arguments, where in week 4 we have a more impactful increase in detection, suggesting that malware will be spawning new processes or using different parameters one month from their first appearance.

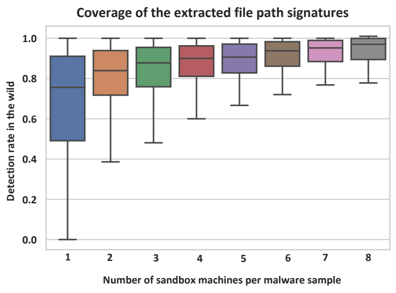
An analyst should re-execute a sample between 3–4 weeks apart. However, having multiple executions in different days provides less useful information about the malware behavior than having different executions on different machines.

5.3 Hunting for the Most Invariant Artifacts

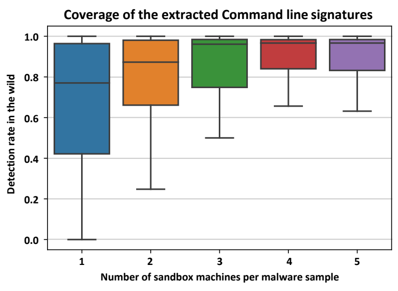
As we showed in previous sections the number of file creations is not a good metric to profile a malware sample due to variability. Similarly, the same file name doesn't appear in



(a)



(b)



(c)

Figure 7: **Detection coverage of tokens obtained by combining multiple execution traces** The detection rate/coverage of file names or extensions reaches the maximum after 3 to 4 machines while for file path we need about 7 machines to capture all the malware tokens.

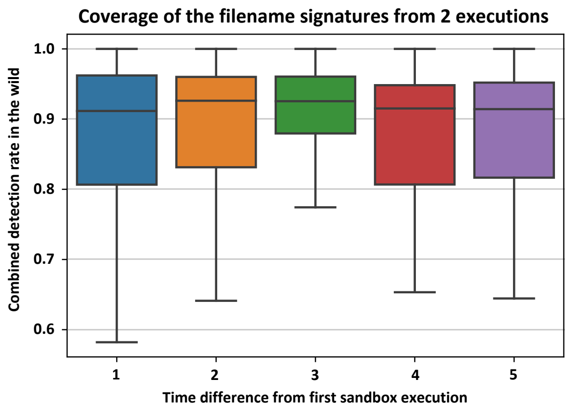


Figure 8: **Total filename signature coverage for re-execution intervals.** A periodic execution of every 3 weeks yields the highest coverage across all malware.

all machines. Using more than 1 file name to profile malware seems like the right choice. While using both variant and invariant features is not going to affect the performance of the detector, we need intuition to be sure we have an invariant in our signature. In this section, we measure the covered machines that individual tokens can detect the malware on. For this we extract the malware tokens in the first week and compare their performance to executions happening in the following weeks, to simulate the scenario where an analyst creates a signature with 1 token and deploys it. We show the average effectiveness of each token. We don't remove the random tokens to show the amount of randomness that an analyst has to deal with for each parameter.

We measure the file name token coverage for file writes. The results show that most of the tokens are random and having more than 1 machine allows the analyst to remove them. We noticed that the tokens with the highest coverage were the extensions, therefore we encourage the analysts to split the file name using the dot(.) delimiter and remove the known benign extensions to obtain highly performing malware file extension signatures. Random tokens happen more often in file names than any other parameter, which means that an analyst should have more than 1 execution to remove the tokens that appear only once.

We noticed that malware tends to write to non-random and non-benign paths. However, there is no clear trend to which subdirectories and on which level are invariant to the malware, therefore, an analyst will need multiple values to construct a signature based on the file path. While we couldn't identify a heuristic to pick the better path tokens we noticed that on average, for all malware, 25% of non-benign subdirectory names (tokens) appear in all the machines. This means that an analyst will achieve a better detection using a subdirectory name to detect malicious file write than the file name, extension or even command line of process creation. Our study reveals a source for constructing high-performance detection rules using file extension tokens, which future generations of malware may no longer posess. We also reveal the success of file path tokens in constructing a malware detection signature.

6 Discussion and Limitations

Impact on State-of-the-Art Solutions. In our paper, we performed an extensive analysis about behavioral variability on malware, concluding that to observe the complete behavior of malware it is necessary to run the malware on several machines repeatedly over time. We conduct two further experiments to illustrate the impact of our results on state-of-the-art solutions.

First we reproduce the experiments conducted in one of the most cited behavioral malware clustering techniques [5]. Such clustering techniques commonly rely on only one execution trace per sample. Note that our goal here is not to call into question the results of the prior work, but to demonstrate the effects of variability in a typical malware-clustering experiment. When evaluating this technique on our data, which consists of several executions of the same malware samples, we observe that one third of the samples exhibit sufficient variability in behavior that their traces were scattered among multiple clusters, thus decreasing the accuracy of mapping samples to the correct family. This suggests that we must be cautious when drawing conclusions from clustering experiments, as the results may be inaccurate if the experiment utilizes a single trace per sample. The details of this experiment can be found in Appendix A.1.

In a second experiment, we assess the impact of behavioral variability on the accuracy of anomaly-detection approaches. In this case, we selected AccessMiner [29], a popular solution for building models based on benign execution alone. It is interesting to note that although variability was not explicitly discussed by the authors, Accessminer correctly accounted for it by including multiple execution of benign software collected from different real-world machine.

Again, we repeated the experiments by following the technique explained in the paper (the details can be found in Appendix A.2), training the AccessMiner model with a progressively increasing number of traces from benign files in our dataset. Our results suggest that behavioral variability of benign programs also impact the detection rate and that only few executions are insufficient to build and accurate model. Moreover, our experiment shows that to improve the accuracy of the models and reduce false alarms, it is necessary to also include lower-reputation and low-prevalence benign files to the dataset. In the original AccessMiner paper, only traces from popular benign files behavior were incorporated into the anomaly detector.

Alternative Solutions to Account for Behavioral Variability. Our findings suggest that the more accurate way to collect information about malware behavior is to record program executions on real end-user machines. However, the main drawback of this solution is that known malware needs to be blocked to guarantee the user security, and thus the data collection is limited to files that other methods cannot classify one way or another.

Other options exist for researchers to account for the behavioral variability of malware. For instance, *Multipath Exploration*, proposed by Moser et al. [36], allows to automatically explore multiple execution paths of the malware binary in the same system. As this method is capable of triggering hidden functionalities, it can replace the need to observe the behavior over different machines and at different points in time. However, this solution is complex and has a very high performance overhead, which makes it unsuitable for large-scale experiments.

Similarly, *Symbolic execution* could be used to trigger unobserved behaviors during malware analysis, such as in the case of time-triggered malware [15]. While this can also help an analyst to tackle the issue of behavior variability, similarly to the multipath exploration solution, symbolic execution is difficult to scale due to the large overhead and the state explosion problems [53].

A more practical solution consists in running the samples on *different VMs*, with different configurations. While still resource intensive, this method has comparably lower overhead than the previous approaches, making it is easier to apply to a large number of samples. As we showed in section 5, we suggest running the malware in at least three (and for better coverage even seven) different VMs to capture significant machine-induced variability. We also suggest the analyst to re-execute the samples at least every three weeks to capture any time-induced variability.

Threats to validity and limitations. Our study carries some limitations due to the nature of the data that was provided by the security vendor. The data was collected from users who have installed the AV product, who might be in general more careful with the security of their computers and, therefore, might be less exposed to attacks. Although we cannot rule out the possibility of selection bias, the large size of the population in our study, the large fraction of malware (9.15% of the unknown samples that could not be classified with other means), and the large spectrum of variability that we observed in the experiments, suggest that our results have a broad applicability.

Our data consists of executions of Windows PE files and therefore, our findings might not apply to the behavior of programs that run on other platforms (Linux [14], Android [55], IoT, etc.). Another unfortunate limitation is that the data does not contain network events. Previous work [43], however, has already shown the existence of a high variability in the network events and discussed its impact on the overall behavior of malware. Since our goal is not to establish a root cause for the behavior variability, the lack of network data does not impact our main findings. We expect to actually observe higher variability on network events.

All samples in our dataset were not flagged neither as benign nor malicious at the time of their collection. Therefore, the data does not include popular benign programs and malware that can be detected by traditional means (i.e., AV Engines). While this might be seen as a limitation because easier to label programs might not show similar variability on their behavior, the set of samples we analyzed also make our study more unique in its nature. We only analyze those programs that need to be detected by looking at the behavior. In reality, samples that can be identified simply by other means, such as static signatures, do not require a behavioral analysis in the first place. Even if our measurement does not capture the variability of those samples, the impact on behavioral detection would have been irrelevant. Moreover, since our goal is to study variations in the runtime behavior, the analysis can only be performed if a sample is executed multiple times, both in the same environment and across a different set of machines. Therefore, polymorphic samples (in which each SHA-256 hash is only observed once) cannot be included in our analysis.

A recent work [46] has shown that for the vast majority of malware samples its impossible to identify a family name, owing to the use of generic signatures and to inconsistencies among the AV labels. Our clustering experiment provides further insight into this challenge. As we were unable to find a family name for most of the samples in our dataset, we did not study the behavior variability across malware families.

When measuring the time variability, some actions may not re-occur. For example, the malware might not recreate files already created in the previous runs, resulting in a significant number of missing events in following runs. However, our results show that during the re-executions of the same sample we often observe *new* events, thus confirming the existence of variability over time.

Finally, our study might have missed malware that can compromise the kernel of the operating system to evade the AV data collection component. This is common to all studies performed on AV telemetry, and since we do not have control over the execution environment we cannot verify the extent of this problem.

7 Related Work

Many prior works explore malware behavior and evolution over time [4, 8, 28] as well as their effects on the accuracy of malware detectors [19, 38]. Our work is also inspired by prior work that establish differences in malware behavior across different sandbox [6, 20, 22] or on behavior that remains dormant [9, 13, 25, 47]. Prudent practices have been proposed when dealing with behavioral data, such as reporting on the exact OS version used for the analysis, which is assumed to affect the observed malicious behavior [43]. Some effort has been made by Lindorfer et al. to detect the existence of one of the factors that affect the behavior of malware: the environmental bias [31]. Our work does not aim at *detecting* malware that show such biases, but instead focus on measuring which parts of the behavior are more prone to environment sensitivity and to which extent. We also differ from this paper, since we are not trying to establish a causal relationship for our results. Pendlebury et al. show that time is another factor affecting the behavior of malware, which they observe through the deteriorating performance of a behavioral classifier [38]. We also measure the effect of time, but look at changes in the behavior instead of at the precision of a classifier.

Finally, while a large body of research has been dedicated to the construction of complex detection models (such as ML classifiers [5, 11, 24, 54]), in but our work we focus on simple token-based rules like those used by SIEM systems [44] and other rule-based detection models [10, 51], because these tokens are the building blocks for more elaborate signatures.

8 Conclusions

It has been known, for over a decade, that malware samples can change their behavior on different hosts and at different points in time, but no study has yet measured this variability in the real world. In this paper, we report the first analysis of malware, PUP and benign-sample behavior in the wild, using execution traces collected from 5.4M real hosts from around the world. We show that malware exhibits more variability than benign samples, In particular, we find that, for at least 50% of the malware, 30% of the actions observed in an execution will not appear in other machines. While there is a lower variability in benign actions, the parameters of these actions are often different. In fact most of the parameters (except for file extension) for all the 3 classes of programs have few values in common across machines. We further show that, for malware that can still execute 2 or more weeks from their first appearance, the variability is lower and so is their detection rate. We then assess the prevalence of invariant parameter tokens that are commonly used to derive behavior based signatures for malware. Even though action parameters that appear in every machine execution are uncommon in malware—50% of the malware samples have only 8% of parameters in common across all executions—we show that we can use 3 to 7 machines to collect parameter tokens that appear in more than 90% of the executions. Our results also suggest that analysts should re-execute the malware samples 3 weeks after first receiving them to update their behavior models. The findings have important implications for malware analysts and sandbox operators, and they emphasize the unique insights that we can gain by monitoring malware behavior at scale, on real hosts.

Acknowledgements

We thank Sandeep Bhatkar and Omer Yampel for helping us understand the behavior data, the anonymous reviewers and SangHyun Hong for their constructive feedback. This research was partially supported by A. James & Alice B. Clark Foundation, the US Department of Defense, and by the European Research Council (ERC) under the Horizon 2020 research and innovation program (grant agreement No 771844 – BitCrumbs).

References

[1] Anubis. http://anubis.cs.ucsb.edu.

[2] Norman sandbox. http://www.norman.com/.

[3] Yara. https://virustotal.github.io/yara/.

[4] Abu Rajab, M., Zarfoss, J., Monrose, F., and Terzis, A. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC* (New York, New York, USA, 2006), ACM Press, pp. 41–52.

[5] Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F., and Nazario, J. Automated Classification and Analysis of Internet Malware. *Recent Advances in Intrusion Detection* (2007), 178–197.

[6] Balzarotti, D., Cova, M., Karlberger, C., Kruegel, C., Kirda, E., and Vigna, G. Efficient Detection of Split Personalities in Malware. *NDSS* (apr 2010).

[7] Bayer, U., Comparetti, P., Hlauschek, C., Kruegel, C., and Kirda, E. Scalable, behavior-based malware clustering. In *Network and Distributed System Security Symposium (NDSS)* (2009).

[8] Bayer, U., Habibi, I., Balzarotti, D., Kirda, E., and Kruegel, C. A View on Current Malware Behaviors. In *LEET* (2009).

[9] Brumley, D., Hartwig, C., Liang, Z., Newsome, J., Song, D., and Yin, H. *Automatically Identifying Trigger-based Behavior in Malware*. Springer US, Boston, MA, 2008, pp. 65–88.

[10] Canali, D., Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., and Kirda, E. A quantitative study of accuracy in system call-based malware detection. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis - ISSTA 2012* (New York, New York, USA, 2012), ACM Press, p. 122.

[11] Christodorescu, M., Jha, S., and Kruegel, C. Mining specifications of malicious behavior. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering - ESEC-FSE '07* (New York, New York, USA, 2007), ACM Press, p. 5.

[12] Christodorescu, M., Jha, S., Seshia, S. A., Song, D., and Bryant, R. E. Semantics-aware malware detection. In *Proceedings - IEEE Symposium on Security and Privacy* (2005), IEEE, pp. 32–46.

[13] Comparetti, P. M., Salvaneschi, G., Kirda, E., Kolbitsch, C., Kruegel, C., and Zanero, S. Identifying Dormant Functionality in Malware Programs. In *2010 IEEE Symposium on Security and Privacy* (2010), IEEE, pp. 61–76.

[14] Cozzi, E., Graziano, M., Fratantonio, Y., and Balzarotti, D. Understanding linux malware. In *2018 IEEE symposium on security and privacy (SP)* (2018), IEEE, pp. 161–175.

[15] Crandall, J. R., Wassermann, G., de Oliveira, D. A., Su, Z., Wu, S. F., and Chong, F. T. Temporal search: Detecting hidden malware timebombs with virtual machines. *ACM SIGOPS Operating Systems Review 40*, 5 (2006), 25–36.

[16] David, O. E., and Netanyahu, N. S. DeepSign: Deep learning for automatic malware signature generation and classification. In *Proceedings of the International Joint Conference on Neural Networks* (jul 2015), vol. 2015-Septe, IEEE, pp. 1–8.

[17] Dinaburg, A., Royal, P., Shari, M., and Lee, W. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the ACM Conference on Computer and Communications Security* (New York, New York, USA, 2008), ACM Press, pp. 51–62.

[18] Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R., and Yan, X. Synthesizing Near-Optimal Malware Specifications from Suspicious Behaviors. In *2010 IEEE Symposium on Security and Privacy* (2010), IEEE, pp. 45–60.

[19] Jordaney, R., Holloway, R., Sharad, K., Laboratories, N. E. C., Dash, S. K., Wang, Z., Papini, D., Elettronica, S. A., Nouretdinov, I., and Cavallaro, L. Transcend : Detecting Concept Drift in Malware Classification Models. In *USENIX Security Symposium* (2017), USENIX Association, pp. 625–642.

[20] Kirat, D., and Vigna, G. Malgene: Automatic extraction of malware analysis evasion signature. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2015), CCS '15, Association for Computing Machinery, pp. 769–780.

[21] Kirat, D., and Vigna, G. MalGene: Automatic extraction of malware analysis evasion signature. In *Proceedings of the ACM Conference on Computer and Communications Security* (New York, New York, USA, 2015), vol. 2015-Octob, ACM Press, pp. 769–780.

[22] Kirat, D., Vigna, G., and Kruegel, C. BareCloud: Bare-metal Analysis-based Evasive Malware Detection. In *23rd USENIX Security Symposium (USENIX Security 14)* (2014).

[23] Kirda, E., Kruegel, C., Banks, G., Vigna, G., and Kemmerer, R. Behavior-based spyware detection. In *Usenix Security Symposium* (2006), p. 694.

[24] Kolbitsch, C., Comparetti, P. M., Kruegel, C., Kirda, E., Zhou, X., and Wang, X. Effective and Efficient Malware Detection at the End Host. In *Presented as part of the 18th USENIX Security Symposium (USENIX Security 09)* (Montreal, Canada, 2009), USENIX.

[25] Kolbitsch, C., Kirda, E., and Kruegel, C. The power of procrastination: Detection and mitigation of execution-stalling malicious code. In *Proceedings of the ACM Conference on Computer and Communications Security* (New York, New York, USA, 2011), ACM Press, pp. 285–296.

[26] Konno, H., Shirakawa, H., and Yamazaki, H. A mean-absolute deviation-skewness portfolio optimization model. *Annals of Operations Research 45*, 1 (1993), 205–220.

[27] Kotzias, P., Bilge, L., Vervier, P.-A., and Caballero, J. Mind Your Own Business: A Longitudinal Study of Threats and Vulnerabilities in Enterprises. In *Network and Distributed System Security Symposium (NDSS)* (2019).

[28] Kwon, B. J., Mondal, J., Jang, J., Bilge, L., and Dumitraş, T. The dropper effect: Insights into malware distribution with downloader graph analytics. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 1118–1129.

[29] Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M., and Kirda, E. Accessminer: Using system-centric models for malware protection. In *Proceedings of the 17th ACM Conference on Computer and Communications Security* (New York, NY, USA, 2010), CCS '10, Association for Computing Machinery, pp. 399–412.

[30] Leys, C., Ley, C., Klein, O., Bernard, P., and Licata, L. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology 49*, 4 (2013), 764–766.

[31] Lindorfer, M., Kolbitsch, C., and Milani Comparetti, P. Detecting environment-sensitive malware. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2011), vol. 6961 LNCS, Springer, Berlin, Heidelberg, pp. 338–357.

[32] Liu, L., Chen, S., Yan, G., and Zhang, Z. Bottracer: Execution-based bot-like malware detection. In *International Conference on Information Security* (2008), Springer, pp. 97–113.

[33] Mann, H. B., and Whitney, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[34] Martignoni, L., Stinson, E., Fredrikson, M., Jha, S., and Mitchell, J. C. A layered architecture for detecting malicious behaviors. In *International Workshop on Recent Advances in Intrusion Detection* (2008), Springer, pp. 78–97.

[35] Miramirkhani, N., Appini, M. P., Nikiforakis, N., and Polychronakis, M. Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts. In *2017 IEEE Symposium on Security and Privacy (SP)* (2017), IEEE, pp. 1009–1024.

[36] Moser, A., Kruegel, C., and Kirda, E. Exploring Multiple Execution Paths for Malware Analysis. In *2007 IEEE Symposium on Security and Privacy (SP '07)* (may 2007), IEEE, pp. 231–245.

[37] Moser, A., Kruegel, C., and Kirda, E. Limits of Static Analysis for Malware Detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)* (dec 2007), IEEE, pp. 421–430.

[38] Pendlebury, F., Pierazzi, F., Jordaney, R., Kinder, J., and Cavallaro, L. {TESSERACT}: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium 2019)* (2019), pp. 729–746.

[39] Praszmo, M. Ramnit – in-depth analysis. https://www.cert.pl/en/news/single/ramnit-in-depth-analysis/.

[40] Rieck, K., Holz, T., Willems, C., Düssel, P., and Laskov, P. Learning and classification of malware behavior. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment* (2008), Springer, pp. 108–125.

[41] Rieck, K., Trinius, P., Willems, C., and Holz, T. Automatic analysis of malware behavior using machine learning. *Journal of Computer Security 19*, 4 (jun 2011), 639–668.

[42] Rossow, C., Dietrich, C., and Bos, H. Large-Scale Analysis of Malware Downloaders. In *DIMVA* (2012), Springer, Berlin, Heidelberg, pp. 42–61.

[43] Rossow, C., Dietrich, C. J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., Bos, H., and Van Steen, M. Prudent practices for designing malware experiments: Status quo and outlook. In *Proceedings - IEEE Symposium on Security and Privacy* (may 2012), IEEE, pp. 65–79.

[44] Roth, F. Generic Signature Format for SIEM Systems. https://github.com/Neo23x0/sigma.

[45] Schreuders, Z. C., Shaw, T., Shan-A-Khuda, M., Ravichandran, G., Keighley, J., and Ordean, M. Security scenario generator (secgen): A framework for generating randomly vulnerable rich-scenario vms for learning computer security and hosting {CTF} events. In *2017 {USENIX} Workshop on Advances in Security Education ({ASE} 17)* (2017).

[46] Sebastian, M., Rivera, R., Kotzias, P., and Caballero, J. Avclass: A tool for massive malware labeling. In *Research in Attacks, Intrusions, and Defenses* (2016).

[47] Sharif, M. I., Lanzi, A., Giffin, J. T., and Lee, W. Impeding malware analysis using conditional code obfuscation. In *NDSS* (2008).

[48] Song, Y., Locasto, M. E., Stavrou, A., Keromytis, A. D., and Stolfo, S. J. On the infeasibility of modeling polymorphic shellcode. In *Proceedings of the 14th ACM conference on Computer and communications security* (2007), pp. 541–551.

[49] ss64. Quotes, Escape Characters, Delimiters - Windows CMD - SS64.com. https://ss64.com/nt/syntax-esc.html.

[50] TrendMicro. TrojanSpy.Win32.GLUPTEBA.A - Threat Encyclopedia - Trend Micro USA. https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/trojanspy.win32.glupteba.a.

[51] Trinius, P., Willems, C., Holz, T., and Rieck, K. A Malware Instruction Set for Behavior-Based Analysis. *Sicherheit Schutz und Zuverlässigkeit SICHERHEIT* (2011).

[52] Willems, C., Holz, T., and Freiling, F. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy 5*, 2 (2007), 32–39.

[53] Yadegari, B., and Debray, S. Symbolic execution of obfuscated code. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security* (2015), pp. 732–744.

[54] Yin, H., Song, D., Egele, M., Kruegel, C., and Kirda, E. Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the ACM Conference on Computer and Communications Security* (New York, New York, USA, 2007), ACM Press, pp. 116–127.

[55] Zhou, Y., and Jiang, X. Dissecting android malware: Characterization and evolution. In *2012 IEEE symposium on security and privacy* (2012), IEEE, pp. 95–109.

A Appendix

A.1 Implications of Variability on Malware Clustering

Dynamic malware clustering [5, 7, 40, 41] aims to identify malware families (or variations withing the same family) by grouping together samples with similar behaviors. These approaches commonly rely on only one execution trace per sample. Therefore, we investigate how the large variability among the traces of each sample could influence the results reported from clustering experiments.

This can be performed by clustering execution traces, and then verifying whether the traces of the same sample are clustered together or they are scattered among multiple clusters. For this experiment, we implemented the clustering technique described by Bailey et al. [5], which also uses similar features to our dataset. As suggested in the paper, we apply their normalized compression distance to our samples, and we utilize the same hierarchical clustering algorithm and the same method to determine the number of clusters. We clustered execution traces from 2,424 malware samples. For each sample we randomly select 4 traces collected in the same week but on different machines; we repeat this step 10 times. We cluster the resulting 9,696 traces, and we obtain 88–105 clusters, of which we pick the median with 93 clusters. To interpret these clusters as families of malware samples with similar behaviors, it is necessary that all executions of a sample fall within its family cluster. In average we found that for 67% of malware samples all 4 executions appeared indeed in the same cluster. However, one third of the samples exhibit sufficient variability in behavior that their traces appear in multiple clusters: 27% fall into 2 clusters, 5% in 3 clusters, and 1% in 4 different clusters. This calls into question the conclusion that the behavior clusters reflect malware families. Because some samples exhibit too much behavior variability to be clustered correctly into families, we must be cautious when drawing conclusions from clustering experiments. Importantly, this threat to validity comes to light when we cluster multiple traces per sample, but remains hidden when using only a single trace per sample.

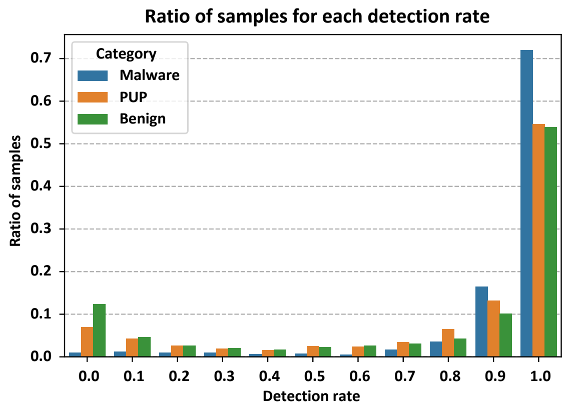
A.2 Impact on Anomaly Detection

One way of detecting malware regardless of their variability is to detect deviations from benign behavior. In this category, Lanzi et al. proposed AccessMiner [29], as system-level anomaly detector based on behavioral traces of benign programs. It is interesting to note that the authors already adopted a technique that accounted for behavioral variability over time and different machine profiles. Similarly to our data, their dataset was also collected from real users but their goal was not to study changes in the application behavior but to obtain a complete picture about how benign files interact with the underlying operating system.

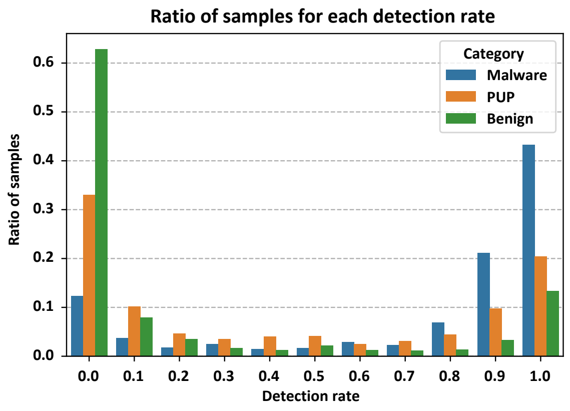
Since in the AccessMiner paper the authors did not discuss how many executions of benign programs are needed to train the anomaly detector, we decided to leverage our data to find an answer to this question such that security companies that opt for anomaly detection rather than malware detection could benefit from our results.

Following the AccessMiner approach, we construct the benign profile by using 90% of the benign executions in our dataset. Remaining 10% is used to measure the false-positive rate. As AccessMiner found file write events to be the most successful in identifying malware, we first build the graph using the file write actions in our dataset. We measure the success of an anomaly based model that relies on only one execution per benign sample (Figure 9a), then the success when all of the executions available to us included (9b). As seen from the figures, a single random benign execution is not sufficient to train an anomaly detector, because it treats most of the executions as anomalies.

The detection rates we obtained from this experiment are lower than the ones reported in the original paper. Concerning that the nature of our data is very different to the benign dataset of AccessMiner this is actually expected. Note that our data consists of unpopular benign applications, whose behavior might be more similar to malicious and unwanted programs. To obtain a better behavioral coverage for benign programs, not only popular benign files such as those used in AccessMiner should be consider but also lower reputation, lower prevalence benign files.



(a) Using 1 random benign execution



(b) Using all benign execution

Figure 9: Amount of samples for the ratio of machines with anomalous file writes.

		25th percentile			Median			75th percentile		
		Mal	**PUP**	**Ben**	**Mal**	**PUP**	**Ben**	**Mal**	**PUP**	**Ben**
File	Path	0.1	–	0.3	0.2	0.5	0.9	0.7	1.0	1.0
	Name	–	–	–	–	0.3	0.5	0.8	1.0	1.0
	Ext.	0.1	0.1	0.5	0.3	0.7	1.0	1.0	1.0	1.0
PE	Path	–	–	–	–	0.5	0.9	1.0	1.0	1.0
	Name	–	–	–	–	0.3	1.0	0.5	1.0	1.0
	Ext.	–	–	–	0.3	1.0	1.0	1.0	1.0	1.0
Reg.Set	Key Path	–	–	–	–	0.3	1.0	0.75	1.0	1.0
	Key Name	–	–	–	–	0.3	1.0	0.81	1.0	1.0
	Value	–	–	–	–	–	0.5	0.33	0.8	1.0
	Dir. Create Path	–	–	–	–	0.6	0.9	0.8	1.0	1.0
	Reg. Create Path	–	–	–	–	–	–	–	0.7	1.0
	Mtx Create Name	–	–	0.1	0.1	0.4	0.8	0.7	1.0	1.0
	New Proc. CMD line	–	–	–	–	–	0.5	0.1	1.0	1.0

Table 5: **Parameter variability for malware, PUP and benign.**
Jaccard index distribution for the top 7 most common actions in our dataset, from week 0 to week 1.

	Median			ratio >0			ratio >2			ratio >5		
	Mal	**PUP**	**Ben**	**Mal**	**PUP**	**Ben**	**Mal**	**PUP**	**Ben**	**Mal**	**PUP**	**Ben**
All actions	39	19	8	99%	95%	83%	96%	91%	68%	92%	77%	57%
File Create	33	2.8	1.8	92%	74%	64%	84%	58%	45%	71%	36%	35%
Mutex Create	6	3	1.3	93%	78%	65%	80%	68%	37%	57%	33%	20%
Registry Set	7.5	3	1	92%	74%	54%	77%	60%	34%	58%	42%	22%
Directory Create	4	1	0	82%	55%	41%	56%	36%	16%	42%	12%	8%
Reg. Key Create	1.8	1	0	71%	52%	36%	39%	34%	18%	7%	16%	10%
PE Create	1.3	0.3	0	76%	50%	35%	28%	29%	18%	6%	13%	12%
Process Load	4	1	0	83%	54%	29%	60%	20%	3%	34%	6%	1%

Table 6: **IQR variability across machines for malware, PUP and benign samples across different machines.** In the last 3 columns we measure the ratio of samples that show a variability greater than 0, 2 and 5 events across machines. The larger the percentage value in those columns, the more samples have IQR greater than the threshold.

Action type	**Mal**	**PUP**	**Ben**
FileCreated	25	60	27
IESecurity	17	12	12
RegistryValueSet	17	10	16
DirectoryCreated	6.7	9	6.3
ProcessLoad	2.7	5	1.9
RegistryKeyCreated	7.8	4	9.9
PECreation	7.9	3.8	15
ProcessInjection	2.9	1.7	2.6
DesktopShortcut	1.4	1.6	1.5
ProcessManipulationInjection	1.2	1.4	1.2
IEHomePage	1.0	1.1	1.6
InternetProxyServer	1.2	1.0	1.4
Others (mean)	≈1.5	≈0.3	≈1.4

Table 7: Average number of actions per execution for executions with at least 1.

Action type	**Mal**	**PUP**	**Ben**
FileCreated	0.74	0.88	0.75
DirectoryCreated	0.47	0.73	0.35
RegistryValueSet	0.56	0.67	0.47
ProcessLoad	0.65	0.65	0.79
PECreation	0.42	0.57	0.29
RegistryKeyCreated	0.31	0.41	0.25
OpenService	0.24	0.29	0.15
DesktopShortcut	0.04	0.14	0.02
CreateService	0.01	0	0.01
KeyloggerShield	0.03	0	0.11
ProcessInjection	0.04	0	0.02
IESecurity	0	0	0
Others (mean)	≈0	0	0

Table 8: Average appearance of an action type across machines.