ML-Based Behavioral Malware Detection Is Far From a Solved Problem

Yigitcan Kaya
UC Santa Barbara
yigitcan@ucsb.edu

Yizheng Chen
Univ. of Maryland, College Park
yzchen@umd.edu

Marcus Botacin
Texas A&M University
botacin@tamu.edu

Shoumik Saha
Univ. of Maryland, College Park
smksaha@umd.edu

Fabio Pierazzi
University College London
f.pierazzi@ucl.ac.uk

Lorenzo Cavallaro
University College London
l.cavallaro@ucl.ac.uk

David Wagner
UC Berkeley
daw@cs.berkeley.edu

Tudor Dumitraş
Univ. of Maryland, College Park
tudor@umd.edu

***Abstract*—Malware detection is a ubiquitous application of Machine Learning (ML) in security. In behavioral malware analysis, the detector relies on features extracted from program execution traces. The research literature has focused on detectors trained with features collected from sandbox environments and evaluated on samples also analyzed in a sandbox. However, in deployment, a malware detector at endpoint hosts often must rely on traces captured from endpoint hosts, not from a sandbox. Thus, there is a gap between the literature and real-world needs.**

We present the first measurement study of the performance of ML-based malware detectors at real-world endpoints. Leveraging a dataset of sandbox traces and a dataset of in-the-wild program traces, we evaluate two scenarios: (i) an endpoint detector trained on sandbox traces (convenient and easy to train), and (ii) an endpoint detector trained on endpoint traces (more challenging to train, since we need to collect telemetry data). We discover a wide gap between the performance as measured using prior evaluation methods in the literature—over 90%—vs. expected performance in endpoint detection—about 20% (scenario (i)) to 50% (scenario (ii)). We characterize the ML challenges that arise in this domain and contribute to this gap, including label noise, distribution shift, and spurious features. Moreover, we show several techniques that achieve 5–30% relative performance improvements over the baselines. Our evidence suggests that applying detectors trained on sandbox data to endpoint detection is challenging. The most promising direction is training detectors directly on endpoint data, which marks a departure from current practice. To promote progress, we will facilitate researchers to perform realistic detector evaluations against our real-world dataset.

I. Introduction

Detection of malware threats is crucial for governments, enterprises, and end users as there are significant (and growing) financial and safety harms of malware infections [1], which has created a $7 Billion industry in 2022 with many players [2]. Malware detection appears to be remarkably effective: industry-standard evaluations of commercial anti-malware products [3], and prior research on malware detection using machine learning (ML) methods [4], [5], routinely report that over 99% of malware samples in a standard corpus can be detected, with very few false positives.

This work has been accepted for publication in the IEEE Conference on Secure and Trustworthy Machine Learning (SaTML). The final version will be available on IEEE Xplore.

Malware detection is often performed at endpoints, where an endpoint security solution [8] monitors host devices to detect threats. In practice, these solutions employ a chain of techniques [9], including static analysis and dynamic analysis. Static methods, such as blocklists and signatures, operate without executing the program. As static methods can readily be bypassed via obfuscation or polymorphism [10], [11], dynamic analysis has become a standard offering [12].

Dynamic analysis relies on observing the execution behaviors of a sample to detect whether it displays malicious behaviors. The typical paradigm for obtaining a behavioral detector is to detonate (execute) a large set of known samples (malware and benign) in a controlled environment (a *sandbox*), collect their execution traces, and learn to separate malicious and benign behaviors. Several ML models have been effective in this task [5], [6], [7], [13]. Unfortunately, endpoint detection models cannot use sandbox traces to decide whether a sample under observation is malicious, as detonation cannot be done in real time, despite advances such as on-premise sandboxes [14]. Consequently, these models must rely on traces observed at real-world hosts to detect malware.

Research has outlined several challenges associated with this task. First, program behaviors are environment-sensitive, and, especially for malware, two traces of the same sample from two different hosts often have little overlap [15]. As a result, an ML model trained on a trace captured on one host might fail to generalize to others. Second, techniques frequently employed to evade sandbox analysis [16], [17] can mislead models trained on sandbox traces into learning spurious features that do not capture actual malware behaviors. Although these challenges have been previously articulated in the research literature [6], [15], [18], [19], we lack an understanding of how they manifest and can be addressed.

Our work conducts the first quantitative study into the efficacy of ML-based methods for endpoint detection to demystify these challenges. We specifically focus on two scenarios: (**i**) an endpoint detector trained using only sandbox traces and (**ii**) an endpoint detector trained using real-world endpoint traces. Existing techniques can conveniently gather large datasets of sandbox traces, making (**i**) a practical and accessible option.

Training data	Test data	Method	TPR @ 1%FPR
Sandbox traces	Sandbox traces	Standard classifier	∼95% [5], [6], [7]
Sandbox traces	Endpoint traces	Standard classifier	∼17% (ours)
Endpoint traces	Endpoint traces	Standard classifier	∼49% (ours)
Sandbox traces	Endpoint traces	Training set resampling + invariant learning	∼22% (ours)
Endpoint traces	Endpoint traces	Soft labels + invariant learning	∼52% (ours)

TABLE I: Performance of ML-based behavioral malware detection under different settings.

On the other hand, collecting endpoint traces is attainable mainly by vendors who receive telemetry data from the wild, as simulating such data at scale in controlled environments (such as sandboxes) is still an open problem [20], [21], [22]. This makes (**ii**) a less accessible but likely a more effective option. Consequently, both scenarios are relevant in practice, which motivates us to study their challenges.

We evaluate three ML approaches [5], [6], [7] that reach 95% true-positive rate (TPR) at 1% false-positive rate (FPR) when evaluated on sandbox traces. We use a dataset from Avlazagaj et al. [15] containing around 1M endpoint traces from 26K samples, recorded on real-world hosts by an anti-malware vendor. The malicious samples in this dataset were undetected by the vendor's defenses at that time and caused real-world infections. Consequently, this data reflects the realistic threats that behavioral detectors must combat. Additionally, we collect a dataset of traces (contemporaneous to our endpoint data) from two sandboxes to realistically study scenario (**i**).

Our initial finding is that ML methods perform poorly in both scenarios (**i**) (∼17% TPR) and (**ii**) (∼49% TPR), in contrast with their excellent performance on sandbox traces (see Table I). We study the low performance in (**i**) from the lens of distribution shift, a problem that plagues ML in many applications [23]. Similarly, the security community has articulated the challenge of concept drift, where the data distribution shifts over time [24]. We attempt to address the differences in data distribution between sandbox vs. endpoint traces using classification with rejection (an existing tool against concept drift [24]) and found that it does not improve the performance enough. Interestingly, endpoint traces of benign samples are also rejected as "drifting" in high proportions, whereas in prior work concept drift has mostly affected malware [25].

Investigating why existing methods perform poorly, we first discovered that endpoint detectors are applied to a different distribution of samples than considered in prior work: they are typically applied only to the hardest-to-classify samples. Past research has trained and evaluated detectors on a corpus of samples from repositories [5], [6], [26] such as MalwareBazaar [27]. However, endpoint malware detection systems employ a pipeline where basic methods (*e.g.*, static signatures) attempt to categorize samples first, and ML-based classifiers are applied only to samples that remain unresolved [9]. Consequently, in the wild, endpoint classifiers are only applied to samples that tend to be harder to classify than an average sample in a standard corpus. Prior research has overlooked this factor, and we find that it causes a significant drop in performance: it reduces the performance of classifiers trained on sandbox traces from 95% (for samples from a standard corpus) to ∼60% TPR (when we adjust the distribution of samples to take into account earlier stages in the pipeline). This shows that prior evaluations have vastly overestimated the effectiveness of behavioral detectors in the wild.

Second, we study the impact of variable program behaviors across different environments. We discovered large differences between a sample's sandbox trace and its endpoint traces. Sandbox traces lack diversity: collecting multiple traces by running a group of related samples, *e.g.*, from the same malware family, in a sandbox produces very similar traces. This introduces spurious features that do not generalize to other environments. In contrast, endpoint traces are diverse, which hurts performance by making a model's predictions on different traces of the same sample inconsistent. Moreover, we provide the first evidence that sandbox-evading malware (40–80% of all malware [17]) biases a model in scenario (**i**) to classify very short traces (often an indication of evasion [28]) as malware. This correlation is spurious as it is absent in endpoint traces; thus, it causes prior evaluations using sandbox traces to overestimate the accuracy of endpoint classifiers.

Characterizing these challenges allows us to explore avenues for performance improvements. To improve the performance on the distribution of difficult-to-classify samples, we use soft labeling (effective against label noise [29]) and more accurate distributions for training. Against variable behaviors, we employ a technique popular in other areas of ML: invariant learning [30]. In particular, we train our models to make consistent predictions on different traces of the same sample, forcing them to learn robust, environment-independent features. Together, these strategies lead to moderate gains, from 17% to 22% TPR in scenario (**i**) and from 49% to 52% in scenario (**ii**) (30% and 5% relative improvement, respectively).

We believe our results should serve as a call to action for the community. Prior research has suggested that it is possible to achieve over 95% TPR, which might leave the impression that progress is saturated and there is not much room for improvement in behavioral malware detection using ML. We show that the reality is different: the problem is not solved, the actual performance on malware in the wild is far worse, and there are major unsolved challenges and significant room for further improvement. Moreover, ML methods that have shown success on standard benchmarks, such as group robustness [31], struggle due to the complexities of this domain. We plan to stimulate progress on this problem by releasing our sandbox dataset and metadata and offering a pipeline that allows researchers to evaluate their behavioral malware detectors against our real-world endpoint data. Our public website (https://malwaredetectionthewild.github.io/)

includes the details on this evaluation pipeline and data release.

Contributions. (I) We measure the performance discrepancy of ML-based malware detectors between sandbox-only and endpoint settings. **(II)** We characterize the ML challenges, such as distribution shifts, behind this discrepancy. **(III)** We explore ML methods to improve the endpoint performance.

II. Background and Related Work

Dynamic Malware Analysis. Most work in dynamic analysis focuses on analyzing the behaviors of a sample detonated in controlled environments, such as sandboxes [32], [33]. As dynamic analysis has become a staple, malware has started including checks that suppress malicious activities if the environment is fingerprinted as a sandbox, known as evasive malware [34]. Although many strategies have been developed to prevent fingerprinting [35], this is still an ongoing arms race [21]. Even the methods that analyze samples in *bare metal* environments [16], [18] struggle to prevent evasion [21]. To our knowledge, we have taken the first step toward measuring the implications of evasion for ML-based detectors at endpoints. A recent large-scale measurement study over variable program behaviors in the wild has found that a given malware sample can behave significantly differently across time and in different real-world hosts [15]. Our work connects to this line of work as we are interested in quantifying the impact of these challenges, such as evasion or variability, on a malware detector deployed for endpoint detection at hosts in the wild.

ML for Malware Detection. ML-based methods are widely used in research and practice with static [4] and behavioral features [36]. In behavioral detection, methods that treat a program's execution trace as a text document (a sequence of tokens) and adapt existing ML architectures have shown promise [5], [6], [7]. Most of these methods are trained and tested on the traces from a pre-configured sandbox for samples collected from public repositories (see [37] for the popular ones). We aim to understand the implications of these practices and the efficacy of popular ML approaches for detecting malware with behavioral features in real-world hosts.

Distribution Shifts in ML. Distribution shift occurs when a model's training and testing distributions have significant differences, which hurts the performance [23]. For example, as new malware variants emerge and old ones disappear over time, the performance of a detector that has not been kept up-to-date will deteriorate, known as concept drift [38]. An effective way to deal with concept drift is deploying drift detectors to reject samples that would have been misclassified and training on them later to update the model [24], [25], [39]. We deploy a drift detector to characterize how the distribution shifts in our problem differ from concept drift. The ML community has also proposed other ideas to tackle distribution shifts in different contexts, such as domain invariant features [40], [41], distributionally robust optimization [31], or continuous learning [39]. We adapt some of these ideas, such as invariant learning, to assess their benefits in our problem.

Limitations of ML for Security. Research suggests that popular ML methods have many pitfalls when applied to security tasks [42]. For example, in malware detection, the success of many methods has been overestimated due to impossible time splits of training and testing sets [38]. Another line of work focuses on how ML methods successful in lab-only evaluations break down in the real world due to distribution shifts, *e.g.*, in the context of network anomaly detection [43], website fingerprinting [44] or malware detection [45], [46]. In these contexts, models are known to learn spurious features, such as specific IP addresses [43], that are artifacts of the experimental setup and do not apply to realistic settings. We investigate the limitations of ML methods specifically in behavioral malware detection when they are trained and evaluated in controlled settings (*e.g.*, using sandboxes) but deployed in the wild.

III. Endpoint Malware Detection

Terminology. A *sample* is an executable program identified by its unique SHA-256 hash. A *trace* is a sequence of behavioral actions (such as file accesses or process creations) performed by a sample when it is executed in a computing environment. We refer to an in-the-wild endpoint environment as a *host*. Our work focuses on Windows hosts and executable samples. In our endpoint dataset, the anti-malware system in each host recorded the traces of samples executed by the host that were not determined to be benign or malicious at an earlier stage of the detection pipeline. Each sample can have multiple traces recorded at multiple hosts at different times. A sandbox (**SB**) is a synthetic environment that provides tools for analyzing samples and recording their traces with controlled executions. Our sandbox dataset contains traces collected from two commercial sandboxes: Tencent HABO [47] and VirusTotal [48], which we will refer to as SB1 and SB2, respectively.

Notation. We denote an individual sample in our dataset as P_i and its ground truth label as y_i, where $y_i = 0$ and $y_i = 1$ indicate a benign and malware sample, respectively. If P_i is a malware sample, it is also tagged with a family label s_i that is useful for grouping samples with similar characteristics. A trace of P_i is x_i^j, where j denotes the execution environment, specifically, $j = 0$ refers to the endpoint hosts and $j \in \{1, 2\}$ refers to the sandboxes. As there are multiple endpoint traces per sample, we refer to the k-th one as $x_{i,k}^0$, and the endpoint traces of a sample are enumerated by their timestamps. The timestamp of its earliest observed trace—$x_{i,0}^0$—marks the first time P_i was first seen in the wild, and $\mathbf{x}_{i,(t<h)}^0$ is the set of all endpoint traces recorded within h hours of this.

An ML-based detection model takes a trace x_i of P_i as its input and aims to infer y_i. We split a model f into two parts: an encoder enc and a classifier g. The encoder produces a vector embedding $z_i = \text{enc}(x_i)$ of the input trace and $f(x_i) = g(z_i)$ is the model's predicted probability (*score*) that P_i is malware. The predicted label $\hat{y}_i$ is $\hat{y}_i = 1$ if $f(x_i) \geq \tau$, or $\hat{y}_i = 0$, otherwise. Here, τ is a tunable threshold, where a higher τ reduces false positives in exchange for fewer true positives. In §III-C, we discuss the tuning of τ in more detail.

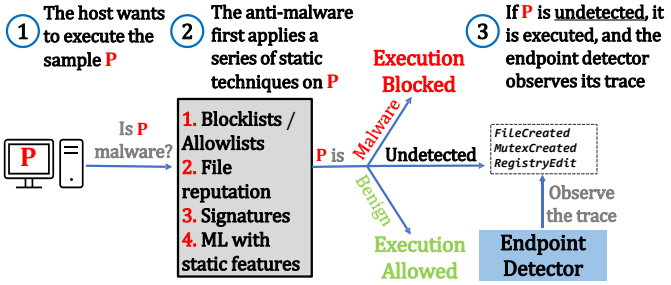


Fig. 1: An overview of endpoint malware detection.

A. Malware Detection Pipeline

Figure 1 provides an overview of the pipeline for malware detection at endpoints. In practice, vendors deploy a chain of techniques [3], [9] to detect if a sample P encountered by a host (*e.g.*, from an e-mail attachment) is malware—step (2) and (3). If one technique in the pipeline fails to classify the sample as malware or benign conclusively, the next one is invoked. First, it deploys high-precision methods (blocklists and signatures), then less precise static ML approaches [4], [49], and finally, dynamic and behavioral analysis. We focus on ML-based behavioral malware detectors deployed in step (3). Static techniques operate without executing the sample and are deployed first as they are more efficient and secure—step (2). This is sufficient for classifying most existing samples [26] either as malware (P is prevented from executing) or benign (P is allowed to execute). If P remains *undetected* after this step, it will be executed if the host demands it, *e.g.*, by clicking on the attachment. If P is malware, executing it leads to a real-world compromise—an *infection*. An endpoint detector is the last line of defense that operates at the host in real-time to identify whether P is malware by consuming the trace resulting from this execution—step (3). If P is detected as malware during this step, anti-malware products often apply remedies such as quarantining, deleting files, or alerting [8].

We identified two issues regarding this pipeline in past research. First, prior work constructed evaluation sets for behavioral malware detectors without considering their position in an end-to-end pipeline. As a result, the challenges in performing behavioral detection on samples that bypassed static methods are unknown. Second, endpoint detectors must classify the sample based on an endpoint trace observed at the end host. It is not feasible to first execute the sample in a sandbox and then classify it based on the sandbox trace, as running in a sandbox (whether in the cloud [50] or on-premise [14]) introduces unacceptable delays. Prior work has only evaluated ML detectors on sandbox traces, but in deployment, the detectors will be applied to endpoint traces, so their actual performance has not been accurately evaluated. We tackle these issues by performing realistic evaluations using our endpoint dataset that contains only the traces of undetected samples—which reached the step (3) in the wild.

Assumptions and Limitations. Security vendors such as VMRay [9], Palo Alto Networks [51], and Kaspersky [52] use multi-tiered solutions that chain static and dynamic analysis in their anti-malware products. We do not have access to these proprietary pipelines. Instead, we consider a streamlined one in Figure 1 based on public documentation by these vendors. Such a pipeline should ideally be evaluated end-to-end, starting with a large pool of samples from the wild, where each component sees only the samples that previous components cannot classify. We are unable to perform such an evaluation as we do not have raw binaries (to extract static features) of our endpoint samples (we could find less than 10% of samples from public sources, mostly malware). Prior malware detection datasets (*e.g.*, SOREL [53]) only share vectorized static features and not the binaries, preventing the simulation of behavioral components. However, we can still realistically isolate and evaluate behavioral ML methods at the pipeline's last stage as our endpoint dataset contains samples that bypassed a real-world product's static components. We recommend future work to collect datasets (with static and dynamic features) that allow the evaluation of the whole pipeline end-to-end or each component according to its position.

Moreover, a pipeline's components constantly evolve in practice. For example, a sample from a new malware family might initially bypass static methods. Eventually, this family can be caught at step (2) with new static signatures or updates to static ML models. Evaluating this requires a dataset with a large temporal span, whereas our endpoint data covers only six months (insufficient for significant changes [25]). Consequently, we perform our evaluation with fixed behavioral ML models that are trained at a particular timestamp and tested on future samples that bypass static methods.

B. Model Training and Evaluation Scenarios

There are two main phases for deploying an ML-based endpoint detector: *training* and *testing*. In the training phase, the vendor collects a set of samples. These samples are often collected from large-scale repositories, either public, such as MalwareBazaar [27], or private [54]. Most prior work in this domain has followed this practice [5], [6], [26], [28], [55]. The vendor can rely on a public platform, such as VirusTotal [48], to label these samples or employ malware analysts in more advanced cases [56]. Most commonly, the collected samples are executed in sandboxes that can conveniently produce large-scale datasets of traces for training. In contrast, collecting a large-scale dataset of endpoint traces from hosts requires a telemetry infrastructure, a permissive user agreement, and a large enough user base—conditions that commercial anti-malware vendors can meet but researchers typically cannot. The collected traces are used to train an ML model that can detect malware from runtime behaviors. During the test phase, the model is deployed locally at each host. Training and testing phases can also be interwoven as vendors continuously update their models on new data [39].

Table II summarizes the detection scenarios, each defined along four dimensions, considered in our work. The SAMPLES dimensions indicate the step of the pipeline training or testing samples originate from: (1)—a broad distribution of all programs hosts encounter—or (3)—a distribution of

programs that remained undetected after (2). The ENV. (short for environment) dimensions encode where the traces are collected—from a sandbox or endpoint hosts.

Deployment Scenario	Training Phase Samples	Training Phase Env.	Testing Phase Samples	Testing Phase Env.
SB->SB	Step (1)	Sandbox	Step (1)	Sandbox
SB->EP	Step (1)	Sandbox	Step (3)	Endpoint
EP->EP	Step (3)	Endpoint	Step (3)	Endpoint

TABLE II: Scenarios for behavioral malware detection.

SB->SB is the scenario most prior work in our domain considers: both training and testing samples are collected from sample repositories and corpora, which represent the distribution of all samples seen in the wild, and these samples are executed in a sandbox [5], [6], [28], [55]. It is already known (and also corroborated by our results) that detectors excel (90%+ TPR) in this scenario [26].

In **SB->EP** and **EP->EP** scenarios, detectors are deployed at (3) in Figure 1, *i.e.*, on the endpoint traces of undetected samples. In **SB->EP**, the detector is trained only on sandbox-based data (same as **SB->SB**) but deployed for endpoint detection. In **EP->EP**, the model is trained on the real-world endpoint traces of undetected samples. This departs from the norm in published research, as most prior detectors have relied on sandbox traces for training. We study this scenario to assess the success of a model trained explicitly for endpoint detection. These scenarios represent a trade-off between practicality (where **SB->EP** is favored) and detection performance (where **EP->EP** is favored).

C. Success Metrics for Endpoint Detection

A conventional detector aims to detect as many malware samples as possible, *i.e.*, to maximize the true-positive rate (TPR), while minimizing the number of benign samples misclassified, *i.e.*, the false-positive rate (FPR). The defender can control the TPR and FPR by tuning the detection threshold τ; a higher τ implies a lower TPR and FPR. Following prior work [57], we evaluate our models regarding their TPR @1% FPR on the test sets (reported as TPR). Additionally, we report the area under the TPR-FPR curve, *i.e.*, the receiver operating characteristic curve, to gauge a model's overall ability to separate malware and benign classes (reported as AUC). Note that a uniform random predictor, regardless of the class proportions in the test set, would achieve 1% TPR @1% FPR and 50% AUC.

Measuring TPR and FPR in **SB->SB** is straightforward as there is a one-to-one mapping from samples to traces. However, in endpoint detection (**SB->EP** and **EP->EP**), there is a one-to-many mapping as multiple hosts may execute the same sample at different times, and these resulting traces often have high variability [15]. For example, a few malware samples have over 400 traces, each corresponding to a real-world infection (see Figure 7a). To obtain a one-to-one mapping, we consider "a representative trace" by randomly selecting one endpoint trace of each sample to create an EP test set on which we measure a model's TPR. We repeat this process 100 times and report the average TPR over these runs as the final TPR. This gives us a fair and unambiguous way to compare performance across scenarios. We refine this metric further based on practical considerations. 67% of traces of an average malware sample (52% for benign) are seen within 24 hours of its first execution at a host. As this ratio drops rapidly after the first day (see Figure 7b) and malware samples die within days [56], we only randomly select from the first-day traces—$\mathbf{x}^0_{i,(t<24)}$—unless stated otherwise.

IV. TECHNICAL SETUP

A. Datasets

Dataset	Training Mal.	Training Ben.	Testing Mal.	Testing Ben.
EP	0.5K	16.5K	0.4K	8.1K
EP (Traces)	19.4K	531.6K	9.7K	412.5K
SB1	46.4K	16.7K	31.0K	16.4K
SB2	34.5K	7.8K	18.1K	7.3K
SB1 ∩ SB2	9.9K	5.3K	8.9K	5.5K

TABLE III: A summary of the samples in our datasets.

Endpoint (EP) Dataset. Provided by Avllazagaj et al. [15], we use a dataset of program traces recorded on real Windows hosts of a commercial anti-malware vendor from over 100 countries between January and July 2018. The vendor did not know whether the samples were benign or malicious at the time of execution. The samples were executed by the users, who interacted with them naturally, and the vendor's behavioral component recorded the traces in a last-ditch effort to discover unknown threats. We have reprocessed this dataset for our problem and relabeled it by querying VirusTotal [48] for more accurate labels. Our processed dataset contains around 1M execution traces from 900 malware and 25K benign samples. Each trace includes high-level actions (file, registry, process, and mutex actions) of a sample that is executed until its termination. To our knowledge, this is the only dataset that allows us to evaluate endpoint detection at scale.

Sandbox (SB) Dataset. A realistic study of **SB->EP** requires a sandbox dataset contemporaneous to our EP dataset. Because the sandbox dataset will serve as the training set and the EP dataset as the test set, we must collect samples seen in or before 2018 to respect causality [38]. Moreover, we cannot detonate samples in a sandbox, as it is commonly done [5], [6], [28], [55], as behaviors of old programs today would differ from their original behaviors. This is because malware stops functioning when, for example, its remote infrastructure dies [56] or it starts behaving differently over time [15]. Our solution is collecting traces from VirusTotal [48], where third-party sandbox vendors publicly share behavior reports on samples. To this end, we curate a list of SHA-256 hashes of Windows samples from popular public malware detection corpora, including EMBER [49], and SOREL [53], released in 2017 or 2018. We then query VirusTotal with these hashes to collect their sandbox traces when available. We discard

traces that came more than six months after the sample was first seen to capture close-to-original behaviors. This results in traces from two sandbox vendors for around 110K malware and 40K benign samples. These vendors are well-known in their countries of origin (SB1 is from China, and SB2 is from the USA). There is a class imbalance as vendors on VirusTotal are more inclined to share traces of malware samples.

Trace Standardization. Our traces from three sources (SB1, SB2, and EP) have different formats, contents, and conventions. We first pre-process our sandbox traces to keep only the type of behaviors recorded in our EP traces, such as file, process, mutex, and registry key creations. We then convert all traces to a single standardized format (see §A-A for details). As shown in §V, the resulting sandbox traces allow us to train models with comparable performance to prior work (∼95% TPR and 99% AUC) in the **SB->SB** scenario.

Temporal Splitting and Labeling. We split our datasets into two portions based on the timestamps of the samples (the first-seen dates) to ensure that the train and test samples are temporally disjoint, avoiding a common pitfall in prior work [38], [42]. Samples seen before April 1st, 2018, and their traces are in the *Training* portion, and the samples seen after that are in the *Testing* portion, on which we never train. As labels of samples are known to change mildly over time [58], we make the best effort to assign the training samples historical labels that were available before April 1st, 2018. We could find such historical labels for ∼24% and 100% of the samples in the EP and SB training portions, respectively. For testing samples, we query VirusTotal to obtain the most recent detection reports and label samples detected by over five anti-malware engines as malware, following the advice in [58]. Further, we use AVClass2 [59] to assign family labels to our malware samples based on their detection reports. We tag malware samples that did not receive a family label as `Generic` malware. This methodology ensures that our evaluation is realistic and reflects the conditions the anti-malware vendor had to work under when our endpoint data was collected.

B. Machine Learning Details

We experiment with three deep-learning-based approaches: (**i**) a bag-of-n-grams-based model (**NGR**), (**ii**) a hybrid model that combines convolution and attention (**HYB**), and (**iii**) a self-attention-based sequence model (**ATT**). These approaches cover a wide range of designs proposed by prior work in behavioral malware detection with ML, respectively, MalDy [5], Neurlux [6] and, most recently, Nebula [7] (see §B for details on these models). We address the class imbalance in our training sets by oversampling the underrepresented classes (malware in EP and benign in SB datasets). Note that our goal is *not* to develop a new architecture or feature processing routine but to evaluate the state-of-the-art ones at endpoint malware detection. We make our ML improvements without changing the fundamentals of existing approaches.

Model Selection. In each setting, we train a set of models with a hyper-parameter grid over model capacity (layer widths), learning rate, regularization (dropout), early-stopping, over-sampling factor, and other setting-specific parameters, such as α in §VII. From this set, we report the average performance of the top-N models, selected according to the metric of interest evaluated on a test set, *e.g.*, TPR on SB1, TPR on EP, or AUC on EP. We opt for performing model selection using the test sets (instead of creating separate validation sets) because our EP dataset contains only a small number of malware samples. We set $N = 20$ so that we can still attribute our results to underlying learning approaches rather than having found a lucky set of hyper-parameters using a test set.

V. Endpoint Detection Evaluations

Table IV presents the detection performance of ML models (based on the methodology in §IV-B), using three different data sets (SB1, SB2, and EP) for training and testing, following the scenarios in Table II. Here, for example, **SB1->SB2** and **SB1->EP** refer to the scenarios where the models are trained on the SB1 training set and evaluated on the SB2 and EP testing sets, respectively. See Table XIII for the AUC results.

A. Evaluating the *SB->EP* Scenario

We see wide performance gaps between **SB->SB** and **SB->EP** scenarios. The gap widens when models are trained and selected using the same sandbox, *e.g.*, 95.0% vs. 11.2% for `NGR` trained and selected using SB1. When models are selected using a different sandbox, the gap shrinks, *e.g.*, 94.0% vs. 13.9% for `NGR` trained on SB1 and selected using SB2. The gap is still substantial even when we select the models based on their EP performance: *e.g.*, 93.2% vs. 16.7% for `NGR` trained on SB1 and selected using EP. Although there is a gap between a model's performances on different sandboxes, *e.g.*, **SB1->SB1** vs. **SB1->SB2**, it is smaller than the **SB->EP** gap. Overall, this gap persists regardless of the training sandbox or the model type, hinting at the limitations of sandbox-trained models for endpoint detection.

We observed that the gap widens when model training and selection use traces from the same sandbox. In §C, we perform an experiment showing this is a generally valid observation. We believe this is because a model that performs better on the traces from an unseen sandbox is less likely to have overfitted to features specific to the training sandbox. We discuss such features in more detail in §VII. This implies that studies that rely on a single sandbox for training and evaluation in **SB->SB** [5], [7], [28], [55] are at a higher risk of producing worse models for **SB->EP**. Consequently, we recommend performing model selection in **SB->EP** using traces from a different, unseen sandbox.

Finally, we explore combining traces from two sandboxes (SB1 and SB2) for training. However, this brings only minor improvements for **SB->EP** over training on traces from one sandbox. When we use EP traces for model selection, training on both sandboxes achieves at most 17.5% TPR on the EP test set for `NGR` (11.7% for `HYB`, and 13.9% for `ATT`). In §VII, we show that the diversity between the traces of a sample from two sandboxes is much lower than between its traces from two

Sel. Test Set	Trained on SB1									Trained on SB2									Trained on EP		
	SB1->SB1			SB1->SB2			SB1->EP			SB2->SB1			SB2->SB2			SB2->EP			EP->EP		
	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT
SB1	95.0	94.2	93.2	60.7	53.7	56.7	11.2	8.0	4.5	48.5	31.8	32.7	85.5	59.2	83.7	14.2	7.2	12.3	43.5	34.0	25.7
SB2	94.0	93.1	92.2	70.4	61.2	64.3	13.9	7.1	5.5	17.6	10.0	14.7	92.6	91.4	90.8	7.5	6.3	11.4	43.4	37.6	26.4
EP	93.2	93.1	87.8	63.5	48.0	52.9	16.7	10.2	8.3	34.5	16.6	15.7	79.3	79.5	88.7	**17.3**	10.9	13.1	**49.5**	43.5	42.8

TABLE IV: The performance (TPR%) of three ML approaches (NGR, HYB, ATT) in seven detection scenarios (based on Table II). The models are selected using the test set in each row, and the average TPR of the top 20 models is reported.

endpoint hosts. This lack of diversity makes combining SB1 and SB2 traces ineffective for increasing the EP performance.

The Impact of Trace Standardization. The standardized trace format (see §IV-A) removes features from sandbox traces unavailable in endpoint traces. In §A-B, to understand whether this might hurt the **SB->EP** performance, we experiment with training and testing models on unstandardized sandbox traces. This shows that standardization moderately hurts the in-domain performance (test traces from the training sandbox) while improving generalization to other domains (traces from other sandboxes). Consequently, we believe standardization helps with improving out-of-domain performance in **SB->EP**.

Applying Drift Detection. We study the low performance in **SB->EP** through the lens of *distribution shift*, which occurs if the process generating inputs, *i.e.*, program traces, changes between the training and testing phases. Concept drift is a highly-studied type of distribution shift in malware detection that stems from malware evolution [38]. In past work on malware concept drift, the distribution shift builds up over time in concept drift, *e.g.*, over 50% performance drop in 2 years [25]. In contrast, we find that in **SB->EP**, the shift manifests immediately after the model is deployed.

One way to combat malware concept drift is to use a drift detector (DD) to identify samples that might have drifted from the training distribution [24], [25], [39], [60]. The classifier then rejects such *out-of-distribution* (OOD) samples while accepting the *in-distribution* (ID) ones on which predictions are reliable [24], [60]. We apply drift detection to quantify the distribution shift present in **SB->EP**. We employ a recent technique ReAct [61] to assign an *outlierness* score to each sample, among which the highest-scoring ones are rejected. ReAct rectifies a model's penultimate layer activations, which calibrates the prediction probabilities of a model to be a better indicator of outlierness. We selected ReAact due to its flexible approach, which can be applied to any neural network model.

We experiment with a NGR model trained on the SB1 training set. We tune ReAct to reject $K \in [0, 15]\%$ of the traces in the SB1 test set (*i.e.*, the ID rejection rate) and separately measure for EP and SB2 test sets (**i**) corresponding rejection rates on the malware and benign traces; and (**ii**) the TPR on the accepted traces. Table V presents the results.

Regarding (**i**), both EP and SB2 traces are rejected at much higher rates than the ID samples, *e.g.*, when $K = 10\%$, ReAct rejects 53% and 38% of the malware traces in EP and SB2, respectively. Malware traces in EP have the highest rejection rate, suggesting that they are the most *drifted* from the malware traces in the SB1 training set. That said, benign traces are also likely to be rejected, *e.g.*, when $K = 10\%$, 39% of the EP, and 37% of the SB2 benign traces are rejected. This is a critical distinction from past work on malware concept drift, where malware samples are rejected at much higher rates than benign samples [25]. The following sections aim to illuminate this phenomenon both qualitatively and quantitatively.

Regarding (**ii**), classification with rejection increases the performance for both EP and SB2 test sets. However, this increase is less pronounced for EP, *e.g.*, gaining ∼10% TPR (over the baselines in Table IV) takes rejecting 67% and 22% of malware traces in EP and SB2, respectively. This also distinguishes the distribution shift in **SB->EP** from concept drift; rejection rates must be high for meaningful performance gains due to the magnitude of the shift.

(K) ID Rej%	SB1->EP			SB1->SB2		
	TPR	Mal%	Ben%	TPR	Mal%	Ben%
5%	20.2%	31.1%	20.5%	79.2%	22.0%	21.5%
10%	23.5%	53.2%	39.0%	89.4%	38.2%	36.5%
15%	25.6%	67.7%	46.4%	88.3%	52.4%	53.6%

TABLE V: The TPR of an NGR model in **SB1->SB2** and **SB1->EP** scenarios, with increasing ID rejection rates and the corresponding rejection rates of malw. and benign traces.

B. Evaluating the *EP->EP* Scenario

For our models in **EP->EP**, we select up to 4 EP traces per sample to form our training set. As we will show in §VII, the behavior variability among the real-world traces of a sample [15] impacts the model's predictions. Consequently, including more traces per sample for training provides better coverage of the behaviors in the wild. We find that using more than 4 traces per sample provides diminishing returns, *e.g.*, for 1, 2, 4, and 16 traces per sample, the TPR of NGR is 43.0%, 44.8%, 49.5%, and 49.9%, respectively.

In Table IV, we see a much higher EP performance in **EP->EP** than in **SB->EP**, as expected. Although training and testing sets are from similar distributions in **EP->EP** (as opposed to **SB->EP**), the performance still lags behind the **SB->SB** performance, *e.g.*, 95% vs. 50% TPR and 99% vs. 88% AUC (Table XIII). In the following sections, we shed light on this discrepancy and what makes endpoint detection particularly more challenging than a sandbox-only scenario.

Takeaways: Distribution shift causes ML methods trained on sandbox traces to perform poorly on endpoint traces (70+% TPR drop). Even when trained on endpoint traces, these methods' endpoint performance still cannot reach their sandbox-based performance (∼20% vs. 95% TPR), hinting at the inherent challenges in endpoint detection.

VI. Endpoint Detection Deals With Difficult-To-Classify Samples

As described in §III-A, an endpoint detector operates on samples that cannot be classified statically—a subset of all samples in the wild. The filtering effect this introduces on the distribution of samples relevant to endpoint detection has not been systematically measured before. To illustrate, we examine (based on Table XIV) the top malware families in our EP test set (undetected samples) and the SB test set (the samples from public corpora). Although both datasets include only the samples first seen in 2017-2018, they encode different priors over samples. For example, generic malware (*i.e.*, malware that could not be placed into any known family) covers 23.8% of the EP samples vs. 4.8% of the SB samples. In the EP set, there are families such as `Khalesi` and `Emotet` that were rampant in 2018 [62]. Conversely, in the SB set, we see older families such as `Sivis` and `Upatre` (circa 2013-2014), whose variants still spread to this day. Moreover, in Table XIV, we share top benign sample publishers (extracted from the code-signing certificates when available), indicating a similar distribution difference in benign samples. Overall, the typical distribution of training and testing samples used by prior work in **`SB->SB`** [5], [6], [26] is substantially different than the distribution of testing samples in **`SB->EP`** and **`EP->EP`**.

Borderline Samples and Label Noise. A common practice in related work in malware detection with ML is discarding *borderline* samples on which the anti-malware engines on VirusTotal are not in consensus. Different works use different criteria; for example, they discard a sample if it is detected by more than zero and less than five engines [28], 20 engines [63] or 40 engines [64]. Even works that do not explicitly discard borderline samples may follow a biased data collection process. For example, recent related work has suggested that dynamic features have limited benefits over static features [26]. The benign samples used in this work were from a trusted Windows software repository, and 93% of their malware samples were detected by 20+ engines.

We consider the samples detected by more than zero and less than 20 engines borderline. This results in 27.7% borderline samples in our EP test set (vs. 15.9% in the SB test set). The prior practice would have discarded over a quarter of the samples that a real-world endpoint detector encountered. Next, we will show that this artificially inflates the measured success of a model. As described in §IV-A, we do not discard any samples and label a sample as malware if more than five engines detect it. This threshold poses a trade-off for the ground truth: as it increases, more malware samples may be labeled as benign, and as it decreases, more benign samples may be labeled as malware. We present a case study in §VI-B to demonstrate how this trade-off plays out.

A. The Impact of Sample Distributions

Here, we measure how the distribution of samples in the test set changes a model's measured (and perceived) performance. To this end, we resample our test sets according to some criteria, *e.g.*, following the malware family distribution in the EP test set, and evaluate our models on these new test sets. This simulates different distributions over a model's test samples and allows us to do controlled experiments. We focus on `NGR` models due to their superior EP performance in §V.

EX#	**`SB->SB`** TPR	**`SB->SB`** AUC	**`SB->EP`** TPR	**`SB->EP`** AUC	**`EP->EP`** TPR	**`EP->EP`** AUC
`EX0`	**No Resampling - Orig. Distributions**					
	93.2	99.0	16.7	78.4	49.5	87.5
`EX1`	**Orig. Distributions w/o Generic Malware**					
	95.6	99.5	18.6	81.4	56.8	91.4
`EX2`	**Only Generic Malware**					
	38.4	85.8	9.8	67.8	23.9	74.2
`EX3`	**Discard Borderline Samples**					
	96.7	99.5	22.6	83.9	66.9	94.2
`EX4`	**Malw. Resampled Following EP Test (w/ Generic)**					
	63.2	94.2	16.7	78.4	49.5	87.5
`EX5`	**Malw. Resampled Following EP Test (w/o Generic)**					
	72.1	97.2	18.6	81.4	56.8	91.4

TABLE VI: The impact of sample distributions in the test set on model evaluations. `EX0-5` denote different experiments.

Table VI presents the results of our resampling experiments. We resample the test sets according to five criteria—denoted as `EX0-5` where `EX0` is testing on the original test set of each respective scenario. In `EX1`, leaving out generic malware samples from original test sets leads to a significant boost in performance, whereas keeping only generic malware in `EX2` causes a massive drop. This aligns with prior claims that generic malware is at the borderline and harder to classify [54]. The fact that `Generic` has $\sim 5\times$ higher coverage in the EP test set than in the SB test set (23.8% vs. 4.8%) also demonstrates the filtering effect that funnels more difficult samples to the endpoint detector. Next, in `EX3`, we discard the borderline samples (identified in the previous section) from all test sets, simulating a common prior practice [26], [28], [63]. This yields a significant boost in all scenarios: 5.9% TPR boost in **`SB->EP`** and 17.4% in **`EP->EP`**. In `EX4` and `EX5`, we resample the test sets to match the distribution of malware families in the EP test set, with and without generic malware, respectively. Note that, for **`SB->EP`** and **`EP->EP`**, `EX4` is equivalent to `EX0` and `EX5` is equivalent to `EX1`. We observe a large decrease in the original **`SB->SB`** performance, *e.g.*, from 93.2% TPR to 63.2% in `EX4` and to 72.1% in `EX5`. Despite neither including any generic malware, the **`SB->SB`** performance in `EX5` is significantly lower

than EX1. This suggests that non-generic malware samples relevant to endpoint detection are generally harder to classify correctly, whether based on their sandbox or endpoint traces. Consequently, the performance of prior behavioral detectors may be overestimated as they are evaluated on samples that are not representative of the distribution of samples faced in endpoint detection. Although the distribution of malware samples explains most of the performance loss when ML classifiers are applied to EP traces, an unexplained gap remains (63.2% vs. 49.5%), which we attempt to explain in §VII.

B. Case Studies

Next, we present two case studies illustrating sample distribution challenges in endpoint malware detection.

Difficult Benign Samples. Table XIV lists Microsoft as the top benign publisher in our EP test set, covering 7.8% of all benign samples. This is counterintuitive as Microsoft is a trusted publisher, and our EP dataset only records samples that could not be classified statically as malware or benign. We discovered that almost all (98%) Microsoft samples in our EP test set share the AM_Delta prefix in their filenames, which correspond to periodic patches to Windows Defender. The NGR model in **EP->EP** outputs an average score of 37% on AM_Delta endpoint traces, almost 2.5× of the average score across all benign traces. Note that the score quantifies the model's confidence that the input trace belongs to a malware sample. In **EP->EP**, on a test set containing only AM_Delta samples as the benign samples, the model achieves only 62.6% AUC, compared to 87.5% with all benign samples. Moreover, there are reports [65] about anti-malware software falsely flagging AM_Delta files. A popular configuration repository for Sysmon—a tool to monitor process activities—includes an allow-list rule specifically for AM_Delta [66]. These suggest that AM_Delta samples are difficult to classify correctly as they make sensitive modifications on hosts. In contrast, our SB test set contains only two AM_Delta samples (less than 0.01% of all benign samples). This highlights that an endpoint detector often faces difficult—false positive prone—benign samples in addition to difficult malware families, which is a root cause behind poor performance in **SB->EP** and **EP->EP**.

Label Noise in Borderline Samples. Roblox is a popular game creation platform. Our EP train and test sets contain 19 and 29 samples from Roblox. Despite being from a trusted publisher, there are reports about anti-malware products flagging Roblox samples. Six (31%) of our 19 training samples were labeled as malware because they were detected by over five engines on VirusTotal, whereas none of the 29 test samples were labeled as malware. This causes a model to associate Roblox samples with malware-ness during training, potentially introducing false positives during testing. For example, our NGR model in **EP->EP** outputs an average score of 40% on the endpoint test traces from Roblox samples. This demonstrates how borderline samples such as Roblox can introduce label noise, particularly in endpoint detection, where they are more common.

C. Combating Sample Distribution Challenges

Building on our observations, we propose two strategies to improve the performance in **SB->EP** and **EP->EP**.

Soft-Labeling Against Label Noise in EP->EP. Malware detectors are generally trained using hard binary labels—0 for benign and 1 for malware. However, hard labels are hazardous when noisy as they force the model to overfit to a wrong prediction [67]. The high frequency of borderline, *i.e.*, potentially noisy, samples exacerbates this problem in endpoint detection. Research suggests that *soft labels* can be effective against label noise [29] by preventing the model from getting too confident on noisy labels during training. To improve the **EP->EP** performance, we implement a function to assign soft labels to our borderline training samples.

For a given sample P_i, our function computes its soft label as $y_i = \min\left(d_i^\theta/\beta^\theta, 1\right)$, where d_i is the number of VirusTotal engines that detected P_i. This function outputs $y_i = 0$ when $d_i = 0$ (confident benign), and it saturates at $y_i = 1$ when $d_i \geq \beta$ (confident malware). The hyper-parameter θ determines how fast y_i grows from 0 to 1 as d_i increases. We present three curves this function generates when $\beta = 20$ in Figure 8. We set $\beta = 20$ and $\theta = 0.75$ without careful tuning and train new models on this soft-labeled EP training set. The results in Table VII (*first* segment) show a significant gain for HYB (5.7% higher TPR) and a moderate gain for NGR (1.1% higher TPR). This suggests that HYB is more vulnerable to label noise, which is expected as HYB is a more complex architecture than NGR with a higher capacity for overfitting. These improvements support our intuition regarding label noise and its effect on ML-based detectors. We leave the exploration of advanced methods against label noise, such as semi-supervised learning [68], to future work.

Before				After			
NGR		HYB		NGR		HYB	
TPR	AUC	TPR	AUC	TPR	AUC	TPR	AUC
Soft Labeling (EP->EP)							
49.5	87.5	43.5	86.8	50.6	87.8	49.2	88.2
Resampled Training Set — Uniform (SB1->EP)							
16.7	78.4	10.2	74.1	15.9	77.2	8.6	74.0
Resampled Training Set — EP Training (SB1->EP)							
16.7	78.4	10.2	74.1	20.0	78.6	11.4	74.9

TABLE VII: Our strategies to improve the performance against sample distribution challenges in endpoint detection.

More Accurate Training Distributions in SB->EP. Our models in **SB->EP** were trained on the distribution of samples captured in public corpora. Although this is standard for obtaining the best **SB->SB** performance [6], [7], it is not ideal for **SB->EP** where the test set follows a different distribution. ML models are heavily influenced by the priors encoded in their training sets because the empirical risk minimization (ERM) paradigm minimizes the average loss over all training samples. For example, 10% of the malware in the SB training

set is from the Virut family, which will be prioritized over Emotet, which accounts for 0.3%.

To alleviate this discrepancy in **SB->EP**, we turn our attention to ML research, where *group robustness* techniques [31], [69], [70] have been effective against a similar problem. These techniques aim to train models that perform well on each subgroup (*e.g.*, demographic groups) instead of favoring groups over-represented in the training set. We implement a prior group robustness method [71] by creating an SB training set with an equal number of samples from each malware family. Unfortunately, as presented in Table VII (*second* segment), this uniform sampling strategy ends up hurting the performance in **SB->EP**. We attribute this shortcoming to the large number of groups (families) in our problem, compared to standardized ML benchmarks, over 600 vs. typically less than 10 [69], [72]. Moreover, malware family labels are often incomplete or noisy [59], [73]. For example, in our SB1 training set, there are many noisy malware family tags such as vbkryjetor (13 samples). As a result, the model is forced to balance hundreds of groups, which also might be inaccurately tagged. These results highlight a key difference between real-world security applications and simplified ML benchmarks, on which most solutions are developed and evaluated, raising doubts about whether an *uninformed* training strategy could work.

Finally, we experiment with an *informed* strategy where the **SB->EP** training set is resampled to follow the family distribution in the EP training set. Widely used threat intelligence platforms can guide practitioners about which families to include in training [74]. Table VII (*third* segment) shows that this strategy brings moderate improvements: 3.3% and 1.2% higher TPR for NGR and HYB, respectively. We will use this strategy to train models in the rest of our paper.

Takeaways: The distribution of samples selected evaluate malware detectors must account for the detector's position in a pipeline. Accordingly, creating realistic test sets for ML-based behavioral detectors (*i.e.*, the final step in the pipeline) reveals the overestimated performance in prior evaluations done on broad distributions (63% vs. 93% TPR).

VII. Environment-Sensitive Program Behaviors Hurt Endpoint Detection

We have shown the performance implications of the distribution of samples encountered by endpoint detectors. However, even when we accounted for distribution differences (EX4 in Table VI), the performance in **SB->SB** (63.2% TPR) is higher than the performance in **SB->EP** (16.7% TPR) and in **EP->EP** (49.5% TPR). In this section, we aim to illuminate the remaining performance gap by studying the impact of variable program behaviors on ML-based approaches.

It is known that program behaviors are environment sensitive and, as a result, a sample can exhibit varying behaviors in different environments [15], [16], [34]. Malware samples are more environment-sensitive than benign samples because, for example, they execute only if the host has specific software vulnerabilities [15] or do not execute if they detect they are running in a sandbox [21], [34]. Building on these insights, we aim to characterize the impact of variable program behaviors on ML-based detectors. We focus on the variability stemming from environment differences between sandboxes and endpoint hosts (for the **SB->EP** scenario) and between different endpoint hosts (for the **EP->EP** scenario). We perform our experiments using the improved NGR models from §VI-C.

A. The Diversity Among Endpoint Traces

Table VIII shows how much traces vary when a set of related samples (*e.g.*, from the same malware family) is run in different environments. We aggregate all the traces from three malware families and two benign publishers in our SB1, SB2, and EP datasets. To cover a range of samples, we select Wannacry (ransomware), Emotet (banking trojan), and Khalesi (info-stealer) as the families, and Opera (web browser), Roblox (game platform) as the publishers. We report the average pairwise distances between these traces within an environment, *e.g.*, between the traces from SB1 (*SB1-SB1*), and between two environments, *e.g.*, between the traces from SB1 and EP (*SB1-EP*). We use normalized compression distance, which has been used in similar contexts [15], [75].

	Within Envs.			Between Envs.		
	SB1-SB1	SB2-SB2	EP-EP	SB1-SB2	SB1-EP	SB2-EP
Wcry	0.16	0.11	0.44	0.28	0.67	0.50
Emot	0.48	0.35	0.50	0.55	0.65	0.53
Khal	0.33	0.29	0.48	0.52	0.62	0.57
Oper	0.05	0.07	0.69	0.43	0.70	0.75
Rblx	0.18	0.04	0.63	0.38	0.67	0.69

TABLE VIII: Avg. pairwise distances between traces of samples from three malware families and two benign publishers.

We observe: (**i**) traces from the same sandbox are almost always very similar; (**ii**) traces from two endpoint hosts are dissimilar (also shown in [15]); (**iii**) traces from different sandboxes are dissimilar but much less so than one trace from a sandbox and one trace from an EP host. The observation (**iii**) can be detrimental for **SB->EP**, as the features learned from SB traces might be irrelevant to EP traces, and (**ii**) is potentially detrimental for both scenarios. The dissimilarity between SB and EP traces implies that a model's training set in **SB->EP** follows a different distribution than its test set, explaining the high drift detection in §V-A.

Next, we demonstrate how (**iii**)—the distance between SB traces and EP traces—affects a model in **SB1->EP**. Figure 2 shows that the model is more confident for traces that are more similar to SB traces. Here, each marker represents a group of 30 traces of a malware family from endpoint hosts, and the x-axis quantifies the average distance of these traces to all traces of this family from SB1. The lower this distance, the more similar a group of endpoint traces is to traces of this family from SB1. For example, the model outputs a 90% score for Emotet EP traces with an average distance of 0.60 to Emotet SB1 traces, which drops to 60% for an EP trace with a distance of 0.75. Consequently, the less similar the

input EP trace is to the SB traces in the training set, the worse decisions the model makes and performs poorly in **SB->EP**.

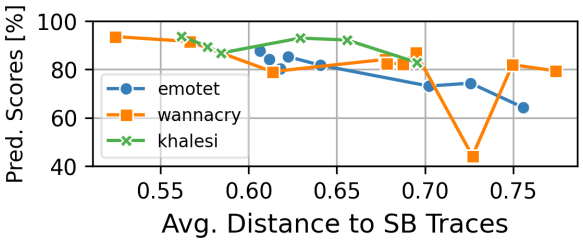


Fig. 2: Relationship between the avg. distance of an input EP trace to the SB traces and the model's output score in **SB->EP**. Each marker represents the avg. of 30 input traces.

In line with our observation (**ii**), research shows that a sample's behaviors vary across hosts due to host-related or external factors [15]. To study how this variability impacts ML, we define $S_{i,h}$ as the set of a model's prediction scores on the EP traces of a sample P_i collected within h hours after P_i was first seen—$S_{i,h} = \{f(x_{i,j}) \mid x_{i,j} \in \mathbf{x}^0_{i,\,(t<h)}\}$.

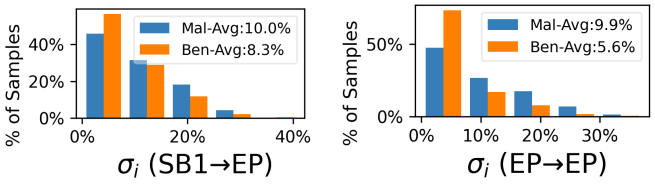


Fig. 3: Prediction score std. deviations on the EP traces of each sample in the EP test set; in **SB1->EP** and **EP->EP**.

Figure 3 presents two histograms of the standard deviations of $S_{i,\,h=24}$ (denoted as σ_i), measured over the samples in the EP test set. With σ_i, we quantify the robustness of a model's predictions to the variability among different EP traces of P_i. First, score variability is higher in **SB->EP**, indicating that a model trained on sandbox traces is more sensitive to behavior variability in the wild. Further, the average σ_i for malware is higher than benign (9.9% vs 5.6% in **EP->EP**), aligning with the finding that malware samples behave more variably across hosts [15]. Score variability is also not uniform across samples; some have nearly zero, and some have over twice the average. This is dictated by the malware type, *e.g.*, botnets execute custom commands, resulting in more variable behaviors [15] and higher score variability. Overall, behavior variability can manifest as score variability and trigger errors (*e.g.*, some traces of a malware sample classified as benign), and ML-based detectors are not naturally robust to it.

The Impact of Sandbox Configurations. Our experiments use sandbox traces obtained from commercial sandboxes with configurations unknown to us, likely optimized to identify malware from its sandbox behaviors. The configuration differences between sandboxes and endpoint hosts (*e.g.*, operating systems, software versions) lead to the large dissimilarity between SB and EP traces we observed. This raises the question of whether a sandbox can be configured to obtain traces that approximate those from real-world hosts. Training on such sandbox traces will likely result in higher performance in **SB->EP**. We acknowledge this as a valid possibility, but it remains an open research question in the current landscape. There is no accepted standard to configure sandboxes in such a manner, and some research suggests that it would be challenging to replicate endpoint systems, *e.g.*, simulating user interactions [35] or diverse real-world configurations [22]. Nonetheless, currently, the research community collects training sets mainly from a single sandbox for ML-based detection solutions [5], [55], [76]. We are unable to collect our own sandbox traces with various configurations to analyze how sandbox configurations affect detection performance, as this would cause a disparity between our endpoint and sandbox data (explained in §IV-A). We highlight this as an important direction for future work.

B. Case Studies

Next, we present two case studies to illustrate the nature of behavior variability and its impact on ML models.

Benign Behavior Variability. `RobloxLauncher` and `OperaPatcher` are two benign samples on the opposite ends of the behavior variability spectrum. `RobloxLauncher` accepts user input through a graphical interface, such as mouse clicks on menu items. We clustered the EP traces of this sample to discover two main execution paths: (**i**) downloading a long list of assets (such as `.mp3` or `.jpeg` files), presumably to update the game; (**ii**) creating temporary Internet files and starting a process, presumably for launching the game. Our **EP->EP** model makes more accurate predictions on the traces in (**i**) than (**ii**) (26% vs 38% average prediction score). In its SB2 trace, the sample downloads a similar list of assets, whereas, in its SB1 trace, it creates only a log file and stops, likely because it could not access the Internet. Neither sandbox triggers the game launch execution path, so we expect a model trained on sandbox traces to struggle to classify endpoint traces that launch the game. Applied to a model trained on SB1 traces, a drift detector (§V-A) consistently rejects the EP traces of this sample. We believe this illustrates a broader challenge: emulating user interactions in sandboxes is challenging [35], causing major differences in the data distribution of sandbox vs endpoint traces from benign samples. Conversely, `OperaPatcher` performs almost the same actions in all its SB1, SB2, and EP traces: creating localization and library files to patch a browser. This sample executing without any user interaction eliminates behavior (and score) variability across traces.

Non-Malicious Malware Traces. In behavioral malware detection, most commonly, the label of a sample (*e.g.*, obtained from VirusTotal) is transferred to all traces from this sample, which are then used for training and testing. This practice implicitly assumes that *all* traces of a malware sample contain some discernible malicious activity. However, this can be problematic in cases where the sample has refused or failed to execute, *e.g.*, because its remote infrastructure is down [56].

To assess this approach, we study the `Wannacry` ransomware family. We select `Wannacry` as it is thoroughly dissected (unlike most families), and its indicators-of-compromise (IOCs) are well-known, allowing us to gauge whether a particular trace is associated with compromise. Using multiple sources, we create a list of IOCs for `Wannacry` and look for them in our SB1, SB2, and EP traces. We split the traces into two sets based on whether they contain any IOC and then measure the average prediction scores of our models on each set. Based on Table XV, we observe (**i**) most traces contain at least one IOC (*e.g.*, 97.7% of the EP traces), (**ii**) the model's average prediction score is lower on the traces with no IOC (*e.g.*, 97% vs. 52% in **`EP->EP`**), which is still significantly higher than the averages on benign traces (13%). The ability to associate malware traces with no IOC with maliciousness is a strength of ML-based methods over IOC-based detectors. Any malware that reaches the execution stage is hazardous, regardless of whether it successfully compromises the host.

C. Spurious Features From Sandboxes

We have shown that sandbox and endpoint traces for a given sample *are* different. Next, we examine *why* they are different and *how* this difference may trigger unreliable predictions from the ML models in the **`SB->EP`** scenario.

Sandbox-Specific Artifacts. We found several sandbox-specific features that are highly prevalent (seen in many traces) and predictive (seen only in malware traces) but occur only in one sandbox and not in the other environments. ML models typically exploit such features to minimize the loss and may overlook other predictive features [77], causing the model to perform poorly on traces collected from a different sandbox or from endpoint hosts. Table IX presents four of these features (called *artifacts*) that we have found in SB1 and SB2 traces. We have identified over 100 such artifacts, most frequently co-occurring in traces. For each artifact, we report its prevalence (**Prv.**) and its malware ratio (**MalR.**) in SB1, SB2, and EP training sets. Prv. is the percentage of samples in which the artifact is present, and MalR. is the percentage of these samples labeled as malware.

File Name	SB1 Traces		SB2 Traces		EP Traces	
	Prv.	MalR.	Prv.	MalR.	Prv.	MalR.
`SogouExp.`	9.0%	100.0%	0.0%	—	0.2%	0.0%
`PersonalB.`	3.3%	100.0%	0.0%	—	0.0%	—
`Spotify`	0.1%	0.0%	3.0%	100%	0.2%	0.0%
`Python`	0.0%	—	3.1%	100%	0.1%	0.0%

TABLE IX: Some strong malware features found in specific sandboxes that do not generalize to other environments.

These artifacts appear to be a result of the specific configuration and particular apps pre-installed on the specific sandbox (see §D). As real-world hosts and other sandboxes do not share the same configuration, this explains why an endpoint classifier trained on traces from one sandbox might fail to generalize to endpoint traces seen in the wild.

The Impact of Sandbox Evasion. Sandbox evasion is a common mechanism malware authors use to make a sample's sandbox trace dissimilar to its real-world traces. If running in a sandbox, evasive malware attempts to avoid analysis or terminate early [21], [78]. In §E, we present a series of quantitative experiments to understand the implications of sandbox evasion for ML-based detectors. We find that an ML-based detector trained on sandbox traces predicts a higher score of maliciousness for the shortest traces. This *length bias* boosts the performance in the **`SB->SB`** scenario, where very short traces are more likely from malware. The same bias, however, hurts the performance in the **`SB->EP`** scenario, where very short traces are not more likely from malware. This makes trace length a spurious correlation learned from sandbox traces that fails to generalize to endpoint traces. Our evidence suggests that this correlation is introduced by evasive malware, which tends to produce short sandbox traces [28].

D. Invariant Learning to Counter Environment-Sensitivity

We have shown that the environment-sensitivity of program behaviors impacts the model. When the classifier exploits features that are specific to the training environment (*e.g.*, a sandbox artifact), its predictions fail to generalize. Table IX suggests that the artifacts of SB1 are absent in SB2, and vice versa, as they are configured sufficiently differently. This highlights an opportunity: learning features that are invariant across two environments (*e.g.*, both sandboxes) might yield better generalization to endpoint detection.

Prior work has studied invariant learning in self-supervised learning, *e.g.*, viewpoint-invariant visual features [79]. We employ the Siamese loss [80], which forces the model to produce similar embeddings (measured by a distance metric) on pairs of related inputs. In our case, each pair consists of two traces of the same sample from different sandboxes (in **`SB->EP`**) or from different EP hosts (in **`EP->EP`**). We aim to make the model invariant to the differences in these pairs (such as sandbox-specific artifacts) to gain robustness to variable behaviors. The critical question is whether invariance across SB traces translates to invariance across EP traces. We represent a pair of traces of a sample P_i as $(x_{i,0}, x_{i,1})$ and the set of all pairs as $\mathbf{D}$; and define the following loss function that is added to the standard ERM loss as a regularizer:

$$\mathcal{L}_{\text{inv}} = \frac{1}{|\mathbf{D}|} \sum_{(x_{i,0},x_{i,1})\in\mathbf{D}} -\cos(\text{enc}(x_{i,0}), \text{enc}(x_{i,1}))$$

$$\mathcal{L}_{\text{final}} = \mathcal{L}_{\text{ERM}} + \alpha\mathcal{L}_{\text{inv}}$$

Here, cos is the cosine similarity, and α controls the intensity of invariance regularization: if set too high, the embeddings might collapse into a single point. In **`SB->EP`**, $\mathbf{D}$ contains the training traces in SB1∩SB2 (Table III), yielding ∼15K pairs. In **`EP->EP`**, we create $\mathbf{D}$ by randomly selecting pairs of traces from each sample in our EP training set (∼88% of the samples have more than one trace). Note that our proposed solution for **`SB->EP`** relies on running a large set of samples in multiple sandboxes. Although this is standard

practice in malware analysis [56], it is typically not done for training ML models for behavioral detection.

Scenario	$\alpha=0$	$\alpha=0.02$	$\alpha=0.08$	$\alpha=0.32$	$\alpha=1.28$
SB1->EP	20.0%	19.1%	20.5%	**21.6%**	20.6%
EP->EP	50.6%	50.6%	50.6%	51.1%	**51.8%**

TABLE X: The impact of $\mathcal{L}_{\text{inv}}$ on endpoint detection TPR.

We present the results in Table X. Here, we start from ($\alpha=0$) the improved NGR models in §VI-C. Both **SB->SB** and **SB->EP** models mildly benefit (1–2% TPR boost) from $\mathcal{L}_{\text{inv}}$, though at different values of α. We hypothesize that some environment-dependent features are still useful, *e.g.*, when a malware family evades one sandbox but not the other, making excessive invariance undesirable. We leave improvements, *e.g.*, selective invariance to preserve useful features, or over more than two sandboxes, to future work.

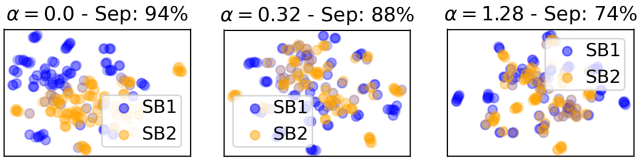


Fig. 4: The model's embeddings with increasing values of α for invariant learning. *Sep* quantifies the separability of the SB1 and SB2 embeddings. t-SNE [81] visualization.

To ensure $\mathcal{L}_{\text{inv}}$ is working as intended, in Figure 4, we compare the embeddings that **SB1->EP** models produce on the testing traces in SB1∩SB2. Increasing α brings the embedding distributions on SB1 and SB2 traces visually closer. Quantitatively, this decreases the accuracy of an SVM classifier in separating these embeddings as coming from SB1 or SB2 (from 94% to 74%). Moreover, Figure 9 presents two histograms of score standard deviations for our invariant models in **SB->EP** and **EP->EP**. In both scenarios, the predictions on EP traces have become less variable (compared to Figure 3). The average standard deviation in **EP->EP** for benign samples decreases significantly (from 5.6% to 0.8%), whereas it remains the same for malware samples. The invariant model has become robust to the variability across the traces of a benign sample, which might not be possible for malware samples whose behaviors vary more. Although we compute $\mathcal{L}_{\text{inv}}$ using only SB1 and SB2 traces in **SB->EP**, the resulting environment-invariance has transferred to endpoint traces, giving us a performance boost. The evidence shows that learning environment-invariant features is promising and can offset the negative impact of behavior variability on ML-based detectors.

Takeaways: An execution environment's specific configuration might introduce artifacts into program traces, hurting behavioral detectors' generalizability to other environments. Learning invariant features over a sample's traces from multiple environments (*e.g.*, two sandboxes) can enhance ML models' robustness to such artifacts, improving their performance in real-world hosts with diverse configurations.

VIII. Discussion and Limitations

Dataset Bias, Size and Label Noise. Our study has some limitations due to our endpoint data. First, this data was collected from hosts that use a single vendor's anti-malware product. Although we cannot rule out selection bias, the fact that these hosts are located in over 100 countries in both enterprise and consumer settings suggests that our results have broad applicability. Second, our data was collected six years ago. This is a common limitation in malware studies as collecting large-scale, up-to-date real-world data is infeasible for researchers [25], [39], [60]. Nevertheless, to our knowledge, our endpoint dataset (originally collected in [15]) is the only dataset of endpoint malware behavior analyzed in the literature. We focus on the gaps between sandbox-based and endpoint detection and not on the specifics of the threats of the time, making our observations still relevant today. Third, our data consists of only Windows hosts and lacks network-related actions. As 95% of malware is aimed at Windows [82] and ML-based detectors can perform well without network actions [6], we do not expect this to affect our findings.

Moreover, our endpoint dataset is much smaller than our sandbox dataset and smaller than what is available to a security vendor, potentially causing us to underestimate the **EP->EP** performance in practice. However, consider Figure 6 where we train models in **EP->EP** using increasing proportions of our endpoint dataset. Here, the performance resulting from using 25%, 50%, and 100% of the data for training is 35%, 41%, and 46% TPR, respectively. This hints that even with larger endpoint dataset sizes, the performance in **EP->EP** is unlikely to catch **SB->SB** due to fundamental challenges we exposed (difficult-to-classify samples and behavior variability).

Finally, for a given sample, labels obtained from VirusTotal (VT) are initially noisy and stabilize after around a year [83]. In our experiments (see §IV-A), we use the oldest label available (*e.g.*, from old VT reports) for the training samples and the labels from VT reports collected four years later (in 2022) for the testing samples. This methodology approximates the realistic conditions for deploying a malware model. To gauge the impact of label noise on our measurements, we compare the labels of training samples from recent VT reports to the labels we used to train our models. In the SB and EP training sets, old and new labels agree on 98% and 97% of the samples, suggesting that the effect of label noise is minimal.

Early Detection of Malware. We study detectors that use the whole execution trace of a sample, and thus can only detect malware once it terminates. Although vendors offer ways to undo the damage from malware after its execution, such as quarantining [8], it is more desirable to catch malware in its tracks, *e.g.*, before a ransomware sample starts encrypting personal files. In that task, a detector has strictly less information available to classify a sample, which makes our results an upper bound on the performance of early malware detection.

Fine-Tuning a Sandbox-Based Model on Endpoint Traces. In §F, we present an experiment where we first train a NGR model only on sandbox traces; we then fine-tune this model

on increasing amounts of data from our EP training set. This hybrid approach simulates a realistic scenario where low-cost sandbox data is supplemented with high-cost endpoint traces from the wild. We implement two strategies: fine-tuning all layers of the model and fine-tuning only the last layer. We find that adding a few EP traces improves performance, and when very few EP traces are available, fine-tuning outperforms training from scratch (3–5% higher TPR).

Attacks Against ML. All ML-based detectors, especially deep-learning-based ones, are subject to adversarial attacks due to their sensitivity to input perturbations [84], [85]. We expect our models will also be vulnerable to such attacks, *e.g.*, an adversary can inject dummy actions into the behaviors of their malware sample designed to fool a model. We consider a defense against these attacks to be out of scope for this work; instead, we aim to show that naturally occurring pressures, such as environment variability, also greatly degrade the performance of ML-based malware detectors in the wild, posing a critical security and trustworthiness challenge in practice.

Promoting Future Research. Our investigation relies on a dataset of endpoint traces from the wild provided by Avllazagaj et al., who collected and analyzed it in their prior work [15]. Using this data, we have pinpointed the salient challenges in ML-based endpoint malware detection. Performing evaluations only on lab-based data can obscure these challenges and produce biased or unreliable solutions inapplicable to the real world [43], [44]. Although its terms of use prevent us from sharing Avllazagaj et al.'s dataset, we built a pipeline that allows the community to perform realistic evaluations of ML-based behavioral malware detectors, aiming to minimize this bias. Anyone may submit to us a pre-trained detector and a feature extractor that converts a trace in the standardized format (§IV-A) into an input to the detector. Upon receiving a submission, we will evaluate it on our endpoint traces dataset and return the submitter a detailed performance report to guide their research. Further, if the submitter wishes, we will publicly list their results and submission details on the leaderboard on our website (https://malwaredetectioninthewild.github.io). To assist participants, we will release the sandbox dataset we collected and the metadata for over 200K samples used in our work. The details of these artifacts can be found in §G. Please refer to our website for technical details, data access requests, and detector submission instructions. This pipeline offers a blueprint for security vendors to drive research without publicly sharing their sensitive data. We hope to spearhead it into more applications of ML for security where evaluation problems frequently hinder real progress.

Implications for Future Work. As described in §III-A, malware detection in practice involves a pipeline of techniques. We believe ignoring this pipeline and treating each component as a standalone solution causes critical evaluation problems. Consider ML-based static malware detectors, an active research area [4], [26], [49]. They are ideally evaluated after filtering out the test samples that can already be detected by the pipeline components preceding static ML, such as existing blocklists, allowlists, and static signatures. However, to our knowledge, this is not an established practice, which might cause an overestimation of real-world performance (similar to our findings in §VI). Our endpoint data allowed us to isolate and realistically evaluate ML-based behavioral detectors, revealing previously unknown challenges. This, however, still falls short of an ideal end-to-end evaluation. Instead of measuring the success of individual components, quantifying the overall effectiveness of a pipeline yields more relevant insights for users. For example, if static components detect 95% of the active malware at a given time and the dynamic components detect 80% of the remaining malware, the overall pipeline would be at 99% TPR. Moreover, the inter-dependence in components introduces poorly understood challenges, *e.g.*, the trade-off between minimizing the pipeline's false positives and classifying more samples statically to reduce infections. Due to our dataset limitations, we are unable to perform such analyses. We recommend that future research approach malware detection holistically, treating it as a pipeline, and collect suitable datasets for performing end-to-end evaluations.

Furthermore, prior work on ML-based behavioral detectors has focused strictly on a sandbox-based scenario where training and testing traces are collected from the same sandbox. This obscures the impact of sandbox configurations on models' robustness when deployed on traces from diverse environments, such as real-world hosts, as shown in §VII. Our work had to rely on traces from two third-party sandboxes with unknown configurations, which prevented us from conducting controlled experiments to study this impact, *e.g.*, to increase the performance in **`SB->EP`**. Instead, we show that invariant learning over traces on these two sandboxes might increase robustness. While sandbox configuration is a well-studied topic in malware analysis [21], [22], [35], its role in generating training sets for ML-based detectors has been overlooked. To address this gap, we recommend that future work tackle realistic scenarios where behavioral models are deployed in unpredictable environments, emphasizing the need for robustness. Developing principled methods for configuring sandboxes for this scenario is a promising research direction.

IX. Conclusion

Behavioral malware detection serves as a last line of defense at endpoint hosts, providing security when all other measures have failed. Malware samples that breach this last line cause real-world infections and harm. A long-standing ambition is to find the best way to detect such samples. Through a systematic exploration of different scenarios, we provide some clarity. State-of-the-art ML-based detectors trained on sandbox execution traces degrade severely when deployed at real-world endpoints. Though training on data from endpoints leads to better performance, it still falls far short of expectations established in prior evaluations. We characterize various challenges ML approaches face in this domain and explore promising techniques targeting them, achieving moderate improvements. Ultimately, this task remains challenging. By exposing evalua-

tion pitfalls and ML shortcomings, we call on the community to seek new solutions to these security-threatening problems.

X. Acknowledgements

This research was supported by the US Intelligence Community Postdoctoral Fellowship (Kaya), the US Department of Defense (Dumitraş), UK EPSRC Grant EP/X015971/1 (Pierazzi), US National Science Foundation Grants CNS-2154873 (Wagner) and CNS-2327427 (Botacin), and generous research awards from Google (Cavallaro and Wagner) and Amazon (Dumitraş and Wagner). We would like to thank VirusTotal for granting us academic access to their platform. Finally, we thank Omer Faruk Akgul and David Acs for their support in the earlier phases of our research and Erin Avllazagaj for providing us access to their program traces in the wild dataset. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the supporting organizations.

References

[1] J. Reddick, "US Treasury: Financial institutions reported $1.2 billion in ransomware losses in 2021," Nov 2022. [Online]. Available: https://therecord.media/us-treasury-financial-institutions-reported-1-2-billion-in-ransomware-losses-in-2021

[2] I. Group, "Malware analysis market: Global industry trends, share, size, growth, opportunity and forecast 2023-2028," 2022. [Online]. Available: https://www.imarcgroup.com/malware-analysis-market

[3] AV-Comparatives, "Malware protection test March 2023," 2023. [Online]. Available: https://www.av-comparatives.org/tests/malware-protection-test-march-2023

[4] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, and C. K. Nicholas, "Malware detection by eating a whole EXE," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[5] E. B. Karbab and M. Debbabi, "MalDy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports," *Digital Investigation*, vol. 28, pp. S77–S87, 2019.

[6] C. Jindal, C. Salls, H. Aghakhani, K. Long, C. Kruegel, and G. Vigna, "Neurlux: dynamic malware analysis without feature engineering," in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 444–455.

[7] D. Trizna, L. Demetrio, B. Biggio, and F. Roli, "Nebula: Self-attention for dynamic malware analysis," 2023.

[8] "Sophos - Endpoint Detection and Response (EDR)," Apr 2024. [Online]. Available: https://www.sophos.com/en-us/cybersecurity-explained/endpoint-detection-and-response

[9] VMRay, "Now, near, deep: The power of multi-layered malware analysis & detection," 2021. [Online]. Available: https://www.vmray.com/cyber-security-blog/now-near-deep-multi-layered-malware-analysis-detection/

[10] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*. IEEE, 2007, pp. 421–430.

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

[12] Kaspersky, "Behavior-based protection." [Online]. Available: https://www.kaspersky.com/enterprise-security/wiki-section/products/behavior-based-protection

[13] E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini, "MAMADROID: Detecting Android Malware by Building Markov Chains of Behavioral Models," in *Annual Symposium on Network and Distributed System Security (NDSS)*, 2017.

[14] "Bitdefender - sandbox analyzer," Apr 2024. [Online]. Available: https://www.bitdefender.com/business/gravityzone-platform/sandbox-analyzer.html

[15] E. Avllazagaj, Z. Zhu, L. Bilge, D. Balzarotti, and T. Dumitraș, "When malware changed its mind: An empirical study of variable program behaviors in the real world," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 3487–3504.

[16] D. Balzarotti, M. Cova, C. Karlberger, E. Kirda, C. Kruegel, and G. Vigna, "Efficient detection of split personalities in malware," in *NDSS*, 2010.

[17] N. Galloro, M. Polino, M. Carminati, A. Continella, and S. Zanero, "A systematical and longitudinal study of evasive behaviors in windows malware," *Computers & Security*, vol. 113, p. 102550, 2022.

[18] D. Kirat, G. Vigna, and C. Kruegel, "BareCloud: Bare-metal analysis-based evasive malware detection," in *23rd USENIX Security Symposium (USENIX Security 14)*, Aug. 2014, pp. 287–301.

[19] D. Kirat and G. Vigna, "Malgene: Automatic extraction of malware analysis evasion signature," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015, pp. 769–780.

[20] Z. C. Schreuders, T. Shaw, M. S.-A. Khuda, G. Ravichandran, J. Keighley, and M. Ordean, "Security Scenario Generator (SecGen): A Framework for Generating Randomly Vulnerable Rich-scenario VMs for Learning Computer Security and Hosting CTF Events," in *ASE@ USENIX Security Symposium*, 2017.

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

[22] A. Mills and P. Legg, "Investigating anti-evasion malware triggers using automated sandbox reconfiguration techniques," *Journal of Cybersecurity and Privacy*, vol. 1, no. 1, pp. 19–39, 2020.

[23] P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, I. Gao *et al.*, "Wilds: A benchmark of in-the-wild distribution shifts," in *International Conference on Machine Learning*, 2021, pp. 5637–5664.

[24] R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, and L. Cavallaro, "Transcend: Detecting concept drift in malware classification models," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017, pp. 625–642.

[25] F. Barbero, F. Pendlebury, F. Pierazzi, and L. Cavallaro, "Transcending Transcend: Revisiting malware classification in the presence of concept drift," in *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2022, pp. 805–823.

[26] S. Dambra, Y. Han, S. Aonzo, P. Kotzias, A. Vitale, J. Caballero, D. Balzarotti, and L. Bilge, "Decoding the secrets of machine learning in malware classification: A deep dive into datasets, feature extraction, and model performance," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 60–74.

[27] "MalwareBazaar," May 2024. [Online]. Available: https://bazaar.abuse.ch/

[28] A. Küchler, A. Mantovani, Y. Han, L. Bilge, and D. Balzarotti, "Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes," in *NDSS*, 2021.

[29] M. Lukasik, S. Bhojanapalli, A. Menon, and S. Kumar, "Does label smoothing mitigate label noise?" in *International Conference on Machine Learning*, 2020, pp. 6448–6458.

[30] H. Zhao, R. T. Des Combes, K. Zhang, and G. Gordon, "On learning invariant representations for domain adaptation," in *International Conference on Machine Learning*, 2019, pp. 7523–7532.

[31] S. Sagawa, A. Raghunathan, P. W. Koh, and P. Liang, "An investigation of why overparameterization exacerbates spurious correlations," in *International Conference on Machine Learning*, 2020, pp. 8346–8356.

[32] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Security & Privacy*, vol. 5, no. 2, pp. 32–39, 2007.

[33] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors," in *Proceedings of the 2nd USENIX Conference on Large-Scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, 2009, p. 8.

[34] M. Lindorfer, C. Kolbitsch, and P. Milani Comparetti, "Detecting environment-sensitive malware," in *Recent Advances in Intrusion Detection: 14th International Symposium, RAID 2011*. Springer, 2011, pp. 338–357.

[35] S. Liu, P. Feng, S. Wang, K. Sun, and J. Cao, "Enhancing malware analysis sandboxes with emulated user behavior," *Computers & Security*, vol. 115, p. 102613, 2022.

[36] W. Huang and J. Stokes, "MtNet: A multi-task neural network for dynamic malware classification," in *Proceedings of 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA 2016)*. Springer, July 2016, pp. 399–418.

[37] "Awesome malware analysis," Apr 2024. [Online]. Available: https://github.com/rshipp/awesome-malware-analysis

[38] F. Pendlebury, F. Pierazzi, R. Jordaney, J. Kinder, and L. Cavallaro, "TESSERACT: Eliminating experimental bias in malware classification across space and time," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 729–746.

[39] Y. Chen, Z. Ding, and D. Wagner, "Continuous learning for android malware detection," in *32nd USENIX Security Symposium (USENIX Security 23)*, Aug. 2023, pp. 1127–1144.

[40] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," in *International Conference on Machine Learning*, 2015, pp. 1180–1189.

[41] B. Gong, K. Grauman, and F. Sha, "Connecting the dots with landmarks: Discriminatively learning domain-invariant features for unsupervised domain adaptation," in *Proceedings of the 30th International Conference on Machine Learning*, 2013, pp. 222–230.

[42] D. Arp, E. Quiring, F. Pendlebury, A. Warnecke, F. Pierazzi, C. Wressnegger, L. Cavallaro, and K. Rieck, "Dos and don'ts of machine learning in computer security," in *31st USENIX Security Symposium (USENIX Security 22)*, Aug. 2022.

[43] A. S. Jacobs, R. Beltiukov, W. Willinger, R. A. Ferreira, A. Gupta, and L. Z. Granville, "AI/ML for Network Security: The Emperor has no Clothes," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1537–1551.

[44] G. Cherubin, R. Jansen, and C. Troncoso, "Online Website Fingerprinting: Evaluating Website Fingerprinting Attacks on Tor in the Real World," in *31st USENIX Security Symposium (USENIX Security 22)*, 2022, pp. 753–770.

[45] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "SoK: The challenges, pitfalls, and perils of using hardware performance counters for security," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 20–38.

[46] H. Aghakhani, F. Gritti, F. Mecca, M. Lindorfer, S. Ortolani, D. Balzarotti, G. Vigna, and C. Kruegel, "When malware is packin'heat; limits of machine learning classifiers based on static analysis features," in *Network and Distributed Systems Security (NDSS) Symposium 2020*, 2020.

[47] "Tencent HABO Sandbox," May 2024. [Online]. Available: https://habo.qq.com/

[48] "Virustotal," Apr 2024. [Online]. Available: https://www.virustotal.com

[49] H. S. Anderson and P. Roth, "EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models," *arXiv preprint arXiv:1804.04637*, 2018.

[50] "Broadcom – Perform Sandbox Analysis in the Cloud," 2024. [Online]. Available: https://techdocs.broadcom.com/us/en/symantec-security-software/web-and-network-security/content-analysis/3-1/about_sandboxing/services_sandboxing_scsb.html

[51] P. A. Networks, "WildFire Administrator's Guide," 2021. [Online]. Available: https://www.niap-ccevs.org/MMO/Product/st_vid11286-agd3.pdf

[52] K. Lab, "Behavior-based protection," 2024. [Online]. Available: https://www.kaspersky.com/enterprise-security/wiki-section/products/behavior-based-protection

[53] R. Harang and E. M. Rudd, "SOREL-20M: A Large Scale Benchmark Dataset for Malicious PE Detection," 2020, arXiv:2012.07634.

[54] X. Ugarte-Pedrero, M. Graziano, and D. Balzarotti, "A close look at a daily dataset of malware samples," *ACM Transactions on Privacy and Security (TOPS)*, vol. 22, no. 1, pp. 1–30, 2019.

[55] B. Miller, A. Kantchelian, M. C. Tschantz, S. Afroz, R. Bachwani, R. Faizullabhoy, L. Huang, V. Shankar, T. Wu, G. Yiu *et al.*, "Reviewer integration and performance measurement for malware detection," in *Detection of Intrusions and Malware, and Vulnerability Assessment: DIMVA 2016*. Springer, 2016, pp. 122–141.

[56] M. Yong Wong, M. Landen, M. Antonakakis, D. M. Blough, E. M. Redmiles, and M. Ahamad, "An inside look into the practice of malware analysis," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3053–3069.

[57] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket," in *NDSS*, vol. 14, 2014, pp. 23–26.

[58] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and modeling the label dynamics of online Anti-Malware engines," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2361–2378.

[59] S. Sebastián and J. Caballero, "AVClass2: Massive Malware Tag Extraction from AV Labels," in *Annual Computer Security Applications Conference*, 2020, pp. 42–53.

[60] L. Yang, W. Guo, Q. Hao, A. Ciptadi, A. Ahmadzadeh, X. Xing, and G. Wang, "CADE: Detecting and explaining concept drift samples for security applications," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 2327–2344.

[61] Y. Sun, C. Guo, and Y. Li, "React: Out-of-distribution detection with rectified activations," *Advances in Neural Information Processing Systems*, vol. 34, pp. 144–157, 2021.

[62] T. H. Team, "The Evolution of Emotet: From Banking Trojan to Threat Distributor," 2018. [Online]. Available: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/evolution-emotet-trojan-distributor

[63] A. Mantovani, S. Aonzo, X. Ugarte-Pedrero, A. Merlo, and D. Balzarotti, "Prevalence and impact of low-entropy packing schemes in the malware ecosystem," in *NDSS 2020*, 2020.

[64] K. Lucas, S. Pai, W. Lin, L. Bauer, M. K. Reiter, and M. Sharif, "Adversarial training for Raw-Binary malware classifiers," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[65] "Is this a legitimate patch from microsoft?" 2021. [Online]. Available: https://answers.microsoft.com/en-us/windows/forum/all/is-this-a-legitimate-patch-from-microsoft/70fd11d4-ce77-43f6-8b09-c7c0fe3e1ba3

[66] ionstorm, "Sysmon ATT&CK Configuration," 2024. [Online]. Available: https://github.com/ion-storm/sysmon-config/blob/94d353f219ce3c62ae01737c0b3d758631328dfa/sysmonconfig-export.xml#L5284

[67] J. Li, Y. Wong, Q. Zhao, and M. S. Kankanhalli, "Learning to learn from noisy labeled data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5051–5059.

[68] X. Wu, W. Guo, J. Yan, B. Coskun, and X. Xing, "From grim reality to practical solution: Malware classification in real-world noise," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2602–2619.

[69] E. Z. Liu, B. Haghgoo, A. S. Chen, A. Raghunathan, P. W. Koh, S. Sagawa, P. Liang, and C. Finn, "Just train twice: Improving group robustness without training group information," in *International Conference on Machine Learning*, 2021, pp. 6781–6792.

[70] M. Zhang and C. Ré, "Contrastive adapters for foundation model group robustness," *arXiv preprint arXiv:2207.07180*, 2022.

[71] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, "Learning imbalanced datasets with label-distribution-aware margin loss," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[72] P. Kirichenko, P. Izmailov, and A. G. Wilson, "Last layer re-training is sufficient for robustness to spurious correlations," in *International Conference on Learning Representations*, 2023.

[73] P. Kotzias, L. Bilge, P.-A. Vervier, and J. Caballero, "Mind your own business: A longitudinal study of threats and vulnerabilities in enterprises," in *NDSS*, 2019.

[74] Cisco, " Cisco Secure Malware Analytics (Threat Grid)," Apr 2024. [Online]. Available: https://www.cisco.com/c/en/us/products/security/threat-grid/index.html

[75] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario, "Automated Classification and Analysis of Internet Malware," in *Recent Advances in Intrusion Detection: RAID 2007*, 2007, pp. 178–197.

[76] D. Trizna, "Quo Vadis: hybrid machine learning meta-model based on contextual and behavioral malware representations," in *Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security*, 2022, pp. 127–136.

[77] M. Pezeshki, O. Kaba, Y. Bengio, A. C. Courville, D. Precup, and G. Lajoie, "Gradient starvation: A learning proclivity in neural networks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 1256–1272, 2021.

[78] T. Roccia, "Evolution of malware sandbox evasion tactics – a retrospective study," Oct 2019. [Online]. Available: https://www.mcafee.com/blogs/other-blogs/mcafee-labs/evolution-of-malware-sandbox-evasion-tactics-a-retrospective-study/

[79] S. Purushwalkam and A. Gupta, "Demystifying contrastive self-supervised learning: Invariances, augmentations and dataset biases," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3407–3418, 2020.
[80] X. Chen and K. He, "Exploring Simple Siamese Representation Learning," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15 750–15 758.
[81] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 11, 2008.
[82] A. VPN, "Over 95% of all new malware threats discovered in 2022 are aimed at Windows," Nov 2022. [Online]. Available: https://atlasvpn.com/blog/over-95-of-all-new-malware-threats-discovered-in-2022-are-aimed-at-windows
[83] S. Zhu, J. Shi, L. Yang, B. Qin, Z. Zhang, L. Song, and G. Wang, "Measuring and Modeling the Label Dynamics of Online Anti-Malware Engines," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2361–2378.
[84] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *International Conference on Learning Representations*, 2014.
[85] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *ESORICS 2017: 22nd European Symposium on Research in Computer Security*, 2017, pp. 62–79.
[86] "Executable process memory analysis," Apr 2024. [Online]. Available: https://www.hybrid-analysis.com/executable-process-memory-analysis
[87] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *International Conference on Machine Learning*, 2009, pp. 1113–1120.
[88] Y. Gorishniy, I. Rubachev, V. Khrulkov, and A. Babenko, "Revisiting deep learning models for tabular data," *Advances in Neural Information Processing Systems*, vol. 34, pp. 18 932–18 943, 2021.

APPENDIX A
TRACE STANDARDIZATION

A. Technical Details

Our EP traces contain high-level information the vendor can seamlessly collect and analyze at endpoint hosts, including file, process, and mutex creations, registry key creations and deletions, and process injections. Sandboxes also record lower-level information (such as memory dumps [86]) as they are less computationally constrained. In a standardized trace, first, we only keep the action types that all formats in our datasets share: file creation, registry key creation and deletion, process creation and injection, and mutex creation. Second, we clean up the strings (*e.g.*, file names) in each trace by removing white spaces, capitalization, punctuation, non-ASCII characters, and so on. Third, we replace certain file and directory names to minimize differences caused by operating system versions or logging conventions. Fourth, we use regular expressions to replace MAC addresses, Windows security and resource identifiers, hashes (SHA-256, SHA-1, MD5), and epoch timestamps with special tokens (*e.g.* `<macaddress>`), to prevent introducing artifacts to our ML models. Fifth, we tokenize the entries in each trace (*e.g.*, split a full file path into its components) and save a trace as a sequence of tokens, following [6]. We will release our sandbox dataset in this standardized format. Moreover, this format enables researchers to submit their detectors to be evaluated against our endpoint dataset (detailed in §VIII) by providing them with an expected input format their detectors should accept. When we train a model, we only keep the top-10K tokens in terms of frequency in the training traces and replace the remaining tokens with special tokens, such as `<rare_file_name>`, again following [6], [7]. This makes the task more suitable for learning by eliminating uninformative tokens and reducing the feature dimensionality.

B. The Performance Impact

We run the following experiment to understand whether our standardization hurts detection performance. We trained `NGR` models on unstandardized SB1 traces, which contain more action types (*e.g.*, file reads, registry reads, and modules loaded) than standardized traces. We also kept the top-25K tokens (instead of the top-10K) from these traces for tokenization. The best model achieves 97.8% TPR on the SB1 test set (up from 95.0% when trained on standardized traces). However, its performance is 53.4% TPR on the SB2 test set (down from 60.7% when trained on standardized traces). This suggests that trace standardization moderately hurts the in-domain performance (on the test traces from the training sandbox) while improving generalization to other domains (to traces from other sandboxes). We hypothesize that standardization reduces overfitting to domain-specific features (discussed in §VII-C), which is desirable in our study as we aim to maximize the performance of a sandbox-based model on endpoint traces. Note that, even after removing some features from sandbox traces, the classifier's performance is still competitive with prior work (*e.g.*, Neurlux [6]) that uses the full sandbox report. This is because different features (*e.g.*, registry reads and registry creations) are already correlated and have redundancy, making each additional feature less effective.

APPENDIX B
FURTHER MACHINE LEARNING DETAILS

`NGR` (based on Maldy [5]). We convert each trace into a list of 2-grams. For example, the file path `a/b/c.jpg` is turned into three 2-grams: `<a/b>`, `<b/c>` and `<c.jpg>`. As this results in an intractable number of unique 2-grams (mostly very rare), we apply the hashing trick [87] that assigns a numerical value up to 2^{14} to each 2-gram. The final feature vector—x—for a trace is a 2^{14}-dimensional vector and each dimension is set to the number of occurrences of the corresponding 2-gram in the trace. We use a ResNet-based architecture [88] to train on these features.

`HYB` (based on Neurlux [6]). We treat a trace as a natural language document and train an attention-convolution-hybrid sequence classification model. This approach eliminates the need for feature engineering (unlike the n-gram approach) and can extract useful features from long sequences thanks to the attention mechanism.

`ATT` (based on Nebula [7]). We treat a trace the same way as `HYB` (a sequence) and train a self-attention transformer-based architecture that is claimed to be robust to heterogeneous information (e.g., different report format).

Overall, these approaches represent an increasing level of complexity, `NGR` being the simplest and most traditional and `ATT` being the most advanced. Although more advanced models seem to perform better in sandbox-based scenarios, we

are interested in whether this trend changes in the endpoint scenario.

APPENDIX C
HOW TO DO MODEL SELECTION IN **SB->EP**

Following our observation in §V-A, in this section, we assess whether finding a better model for **SB->EP** is possible by selecting the models using traces from an unseen sandbox (not used for training). We rank our models (100+ in each setting) based on their SB1, SB2, and EP performances (TPR) and compute Spearman's correlation coefficients between these rankings. Table XI reveals that (except for one setting) (**i**) the rankings based on the training sandbox correlate poorly (sometimes negatively) with EP rankings, (**ii**) rankings based on an unseen sandbox correlate more strongly with EP rankings. For example, for ATT trained on SB1, the rankings based on SB1 and SB2 have 0.03 and 0.45 correlation with the rankings based on EP, respectively. These results support our claim that it yields better results when model selection in **SB->EP** is performed using traces from a different, unseen sandbox that is not used for training.

	Trained on SB1			**Trained on SB2**		
	SB1-SB2	SB1-EP	SB2-EP	SB1-SB2	SB1-EP	SB2-EP
NGR	0.47	0.03	0.45	−0.24	0.39	−0.46
HYB	0.88	0.84	0.75	−0.17	0.59	0.33
ATT	0.56	−0.17	0.31	0.51	0.58	0.35

TABLE XI: Ranking correlations of sandbox-trained models according to their performances (TPR) on different test sets.

APPENDIX D
CASE STUDIES ON SANDBOX-SPECIFIC ARTIFACTS

Here, we dive deeper into the sandbox-specific artifacts we identified in Table IX.

SogouExplorer is a Chinese web browser that exists only in malware traces (100% MalR.) in SB1; whereas it does not exist in any SB2 traces and very few EP traces. The samples that interact with it mainly belong to families such as Sivis and Memery, all tagged as file infectors that attach their code to other programs. Considering that a Chinese vendor developed SB1, we believe they pre-installed this browser on their sandboxes to generate an analysis environment representative of Chinese hosts. This, however, creates features specific to SB1 as samples interact with the programs in the environment. Although this artifact exists in a few endpoint traces from hosts in China, its prevalence is almost zero.

PersonalBankPortal, according to our research, is a program distributed by a Chinese bank to its customers. The samples that inject into this program belong to families such as Tinba and Ramnit, all considered as banking trojans that specifically ex-filtrate banking data. We believe the vendor pre-installs this program to lure malware samples into exhibiting their behaviors. Although this practice is useful for analyzing a sample [56] (and for **SB->SB**), it causes artifacts that are rarely observed in the wild.

Among the artifacts found in SB2, **Spotify** is a popular music streaming service, and **Python** is the interpreter for Python programming language. Both programs are targeted and injected by file infectors, similar to SogouExplorer in SB1. These programs are much less prevalent in endpoint traces than in SB2 traces. We believe the SB2 vendor, based in the US, pre-installs them to create an environment representative of the hosts in the US.

APPENDIX E
THE IMPACT OF SANDBOX EVASION

Recent works have measured that 40-80% of malware uses at least one evasive technique [17] to avoid analysis or terminate early if it is running in a sandbox [21], [78]. Evasion makes sandbox traces dissimilar to endpoint traces. Although sandbox evasion is well understood, its implications for endpoint detection have not been measured.

We explore the impact of sandbox evasion. We use a standard heuristic and treat a sample as evasive if the number of actions it performs is too low [16], [21], [28]. We first find the malware families common between our EP and SB1 test sets (30 total). We then compute the average trace length of each common family using the traces of the samples belonging to it. We count only registry actions, as they can be recorded unambiguously, unlike actions such as process injections, which might have vendor-specific definitions. We then find the length differences between SB1 and EP traces of each family. For example, Wannacry and Gandcrypt have differences of $+11$ and -6, respectively. We then split the families into two sets: the 21 families whose SB traces are longer (*e.g.*, Wannacry) are deemed less likely to be evasive, and the 9 families whose SB traces are shorter (*e.g.*, Gandcrypt) are deemed more likely. The median length differences for the non-evasive and evasive families are $+4.4$ and -4.3, respectively, with a few outliers on both sides, *e.g.*, $+81$ for Vobfus and -85 for Bypassuac. Finally, we measure the TPR of our model in **SB1->SB1** and **SB1->EP** on these two malware sets individually while keeping the benign samples the same.

Families	#Fams	#EP	#SB1	**SB1->EP**	**SB1->SB1**
All	30	169	5.8K	23.1%	86.4%
Evasive	9	49	2.9K	20.4%	89.8%
Non-Eva.	21	120	2.9K	23.3%	83.0%

TABLE XII: The performance (TPR) on malw. families more (*Evasive*) or less (*Non-Eva*) likely to be evading sandboxes.

The trace length of a sample monotonically increases as its execution continues. This means the discrepancies between the execution durations in SB1 and endpoint hosts might confound our measurements. Before presenting our results, we confirm this is unlikely to be the case: 82% of our EP traces are from executions that lasted less than 90 seconds. Although we do not know the exact configuration of SB1, allowing one to two minutes of execution is standard for most sandboxes in practice [28].

Based on the results in Table XII, we observe: (**i**) **`SB1->EP`** performance is higher (by 3%) on non-evasive families than on evasive families; (**ii**) **`SB1->SB1`** performance is significantly higher (by 7%) on evasive families than on non-evasive families. This suggests that the classifier exploits short traces (evasiveness) as a feature, which is beneficial for **`SB->SB`**, though it does not generalize to **`SB->EP`**. However, the non-trivial **`SB1->EP`** performance on evasive families might hint that there are still features useful for endpoint detection in the sandbox traces of evasive samples. Although we are limited to observational data, a controlled study on evasive malware to disentangle these features is a promising direction.

The Trace Length Bias. Building on the previous experiment, we hypothesize that a model in **`SB->EP`** might learn an inverse correlation between trace length and malware-ness, *i.e.*, evasive malware creates short traces, and, therefore, short traces are more likely to be malware. In Figure 5, we present the model's average predicted scores on traces with a certain length in SB1 and EP test sets. For both SB and EP test sets, we also measure the ratio of malware traces of a given length labeled among all traces of that length. For example, if there are 100 total traces of length L and 70 of them are labeled as malware in the ground truth, the malware ratio would be 0.7 for the traces of length L.

These plots support our hypothesis: the model predicts a higher score for the shortest traces in both test sets, *i.e.*, it has a length bias. This bias leads to accurate predictions in the SB set, where very short traces are much more likely to be malware (the malware ratio on the leftmost side of the upper plot is high). However, this bias leads to inaccurate predictions in the EP set, where very short are not more likely to be malware. On the EP traces shorter than 10 actions (∼20% of all EP traces), the model achieves 15% TPR (vs. 20% when all traces are kept). Ultimately, trace length is a spurious correlation learned from sandbox traces that fails to generalize to endpoint traces. Evidence suggests this is introduced by evasive malware that tends to produce short sandbox traces in the same execution time. Methods preventing the model from learning such correlations [72], [77] offer a promising next step.

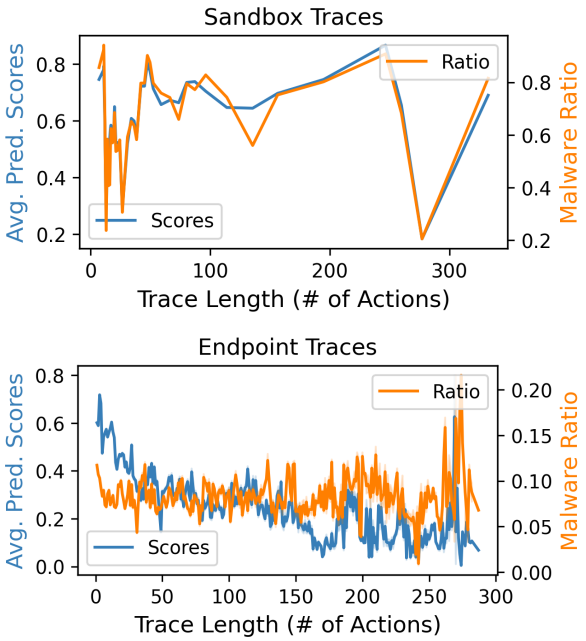


Fig. 5: Comparing the correlations between trace length and malware-ness prediction scores of the model. *Malware ratio* is the ground truth ratio of malware traces of a certain length among all traces of that length in a dataset.

Appendix F
Fine-Tuning a Sandbox-Based Model on Endpoint Traces

For the experiments in Figure 6, we select two endpoint traces per sample and implement two fine-tuning strategies: **(i)** freezing the encoder layers enc of our model and tuning only the classification layer g, and **(ii)** tuning all layers without freezing. We train models on increasing portions of the samples in our EP training set (randomly selected) and average the results over 10 models. We make the following observations. In low-data regimes (below 30% of the EP data), fine-tuning only g outperforms the other options in terms of TPR by ∼3–5%. However, with more data, it starts to perform significantly worse, due to being less flexible in learning from the EP traces. Finally, fine-tuning all layers is generally the worst option, and training from scratch is the best when more EP data is available. These experiments show that starting from a well-performing model in **`SB->EP`** is beneficial in regimes with limited EP data, highlighting a promising direction for future work.

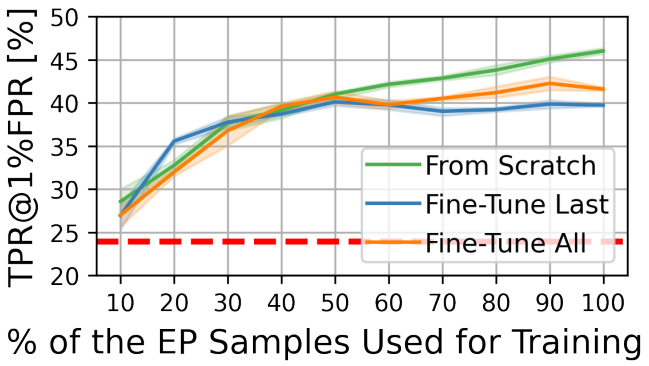


Fig. 6: The results of fine-tuning a sandbox-based model in **SB->EP** on endpoint traces. The dashed line indicates our best model **SB->EP**. Experiments on NGR.

Appendix G
More Details on the Artifact Release

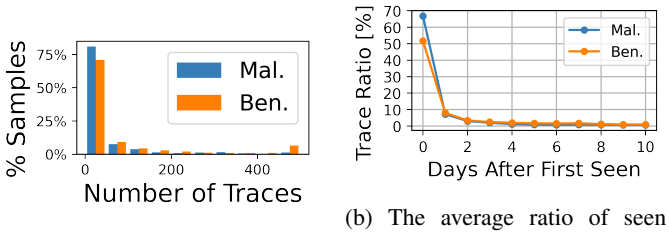
We will release the following artifacts to the community (visit our website for more details https://malwaredetectioninthewild.github.io/):

Sandbox Dataset. We will release the training portion of our sandbox datasets in Table III, stored in the standardized format discussed in §IV-A. We avoid releasing the testing portion publicly to prevent researchers who wish to participate in our realistic evaluation leaderboard from obtaining an impractical advantage by training their detectors on it (will be released if requested in exchange for being excluded from the leaderboard). This will level the playing field for participants and ensure they all have access to the same sandbox data for development. Further, as this data contains traces from two sandboxes, participants can leverage our observations in §V and tune their hyper-parameters on a second sandbox, not seen during training, for improving the endpoint performance or train their models on data from both sandboxes. They can also apply invariant learning techniques, which have shown promising results in §VII-D.

Sample Metadata. We will release the metadata relating to the training samples in all datasets, including their SHA-256 hashes, ground truth labels, family tags (if malware), and first-seen timestamps. Additionally, our metadata also annotates the source of a sample, *e.g.*, EMBER [49], SOREL [53], or our endpoint dataset. This allows researchers to compute realistic priors over malware families for endpoint detection and make use of our findings and improvements in §VI-C. Moreover, they can also create realistic testing distributions over samples using these malware family priors (and avoid the problems we discussed in §VI.

Appendix H
Additional Tables and Figures

Statistics on the Endpoint Traces



(a) The number of traces per sample.

(b) The average ratio of seen traces per sample as a function of time.

Fig. 7: Statistics on the endpoint traces in our dataset.

Soft Labeling Function

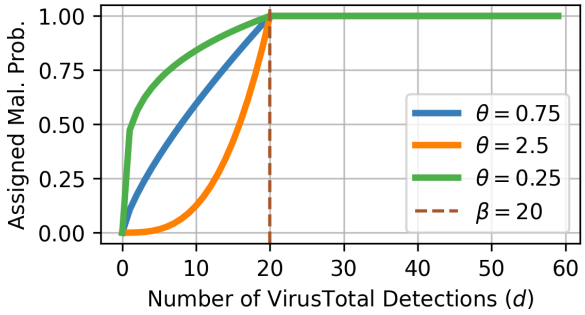


Fig. 8: Our function for assigning probabilistic (soft) labels to each sample based on the number of VirusTotal detections.

Histogram of Prediction Score Standard Deviations on Endpoint Traces of the Same Sample for Models Trained With the Invariance Loss.

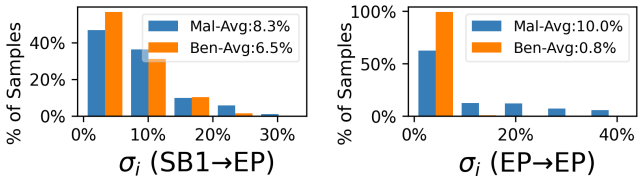


Fig. 9: Prediction score standard deviations on the endpoint traces of each sample in the EP test set; for a model in **SB1->EP** (*left*), and **EP->EP** (*right*).

Sel. Test Set	Trained on SB1									Trained on SB2									Trained on EP		
	SB1->SB1			SB1->SB2			SB1->EP			SB2->SB1			SB2->SB2			SB2->EP			EP->EP		
	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT	NGR	HYB	ATT
SB1	99.1	99.0	98.9	93.8	91.3	92.6	76.1	70.7	72.8	89.5	83.4	89.1	97.7	94.1	96.9	72.0	68.2	70.8	85.6	85.2	84.3
SB2	99.0	98.6	98.8	94.9	93.2	93.5	76.6	71.3	74.4	81.7	71.5	86.4	98.7	98.5	98.2	71.6	66.7	72.7	85.4	85.9	84.8
EP	99.0	98.6	98.7	92.9	91.2	92.9	**78.4**	74.1	74.9	84.4	71.8	88.3	98.0	97.8	98.0	74.6	72.0	73.4	**87.5**	86.8	86.8

TABLE XIII: The performance (AUC%) of three ML approaches (NGR, HYB, ATT) in seven detection scenarios (based on Table II). The models are selected using the test set in each row, and the average AUC of the top 20 models is reported.

Top Malware Families and Benign Publishers

EP Test Set			SB Test Set		
Name	EP%	SB1%	Name	SB1%	EP%
GENERIC	23.8%	4.8%	GENERIC	4.8%	23.8%
Chindo	9.0%	0.0%	Gepys	4.4%	0.0%
Emotet	7.6%	0.5%	Sivis	4.3%	0.3%
Gandcrab	6.0%	3.6%	Flystudio	4.2%	0.3%
Loadmoney	6.0%	0.1%	Upatre	3.8%	0.8%
Khalesi	5.0%	0.3%	Gandcrab	3.6%	6.0%
Installcore	4.5%	0.0%	Shipup	3.5%	0.0%
UNSIGNED	29.6%	47.2%	UNSIGNED	47.2%	28.4%
Microsoft	7.8%	0.4%	Google	4.7%	0.2%
Tencent	2.6%	0.5%	Mozilla	3.7%	0.1%
Qihoo	2.2%	0.2%	Digital R.	3.5%	0.0%
Zoho	1.3%	0.1%	Yandex	2.7%	1.0%
Opera	1.2%	0.1%	ScreenC.	2.2%	0.0%
Yandex	1.0%	2.7%	Zoom	2.0%	0.3%

TABLE XIV: Top malware families (*top*) and benign (*bottom*) publishers in our EP (*left*) and SB (*right*) test sets, along with their shares in each dataset.

Case Study on Wannacry and Its Indicators-Of-Compromise

Trace Type	Ratio			Avg. Pred. Score		
	SB1	SB2	EP	SB1->EP	SB2->EP	EP->EP
Any IOC	89.9%	100%	97.7%	82.0%	75.6%	97.4%
No IOC	10.1%	0.0%	2.3%	77.7%	67.7%	51.8%

TABLE XV: For Wannacry traces in our SB1, SB2, and EP datasets, we first measure the ratio of traces with at least one and no IOC. We then measure the average prediction score of our NGR models in **SB1->EP**, **SB2->EP** and **EP->EP** on these traces.