Mind the Gap: On Bridging the Semantic Gap between Machine Learning and Malware Analysis

Michael R. Smith
Sandia National Laboratories
Albuquerque, New Mexico
msmith4@sandia.gov

Nicholas T. Johnson
Sandia National Laboratories
Albuquerque, New Mexico
nicjohn@sandia.gov

Joe B. Ingram
Sandia National Laboratories
Albuquerque, New Mexico
jbingra@sandia.gov

Armida J. Carbajal
Sandia National Laboratories
Albuquerque, New Mexico
ajcarba@sandia.gov

Bridget I. Haus
USC Viterbi School of Engineering
Los Angeles, California
bhaus@usc.edu

Eva Domschot
New Mexico Tech
Socorro, New Mexico
eva.domschot@student.nmt.edu

Ramyaa Ramyaa
New Mexico Tech
Socorro, New Mexico
ramyaa@cs.nmt.edu

Christopher C. Lamb
Sandia National Laboratories
Albuquerque, New Mexico
cclamb@sandia.gov

Stephen J. Verzi
Sandia National Laboratories
Albuquerque, New Mexico
sjverzi@sandia.gov

W. Philip Kegelmeyer
Sandia National Laboratories
Albuquerque, New Mexico
wpk@sandia.gov

ABSTRACT

Machine learning (ML) techniques are being used to detect increasing amounts of malware and variants. Despite successful applications of ML, we hypothesize that the full potential of ML is not realized in malware analysis (MA) due to a semantic gap between the ML and MA communities—as demonstrated in the data that is used. Due in part to the available data, ML has primarily focused on detection whereas MA is also interested in identifying behaviors. We review existing open-source malware datasets used in ML and find a lack of behavioral information that could facilitate stronger impact by ML in MA. As a first step in bridging this gap, we label existing data with behavioral information using open-source MA reports—1) altering the analysis from identifying malware to identifying behaviors, 2) aligning ML better with MA, and 3) allowing ML models to generalize to novel malware in a zero/few-shot learning manner. We classify the behavior of a malware family not seen during training using transfer learning from a state-of-the-art model for malware family classification and achieve 57% - 84% accuracy on behavioral identification but fail to outperform the baseline set by a majority class predictor. This highlights opportunities for improvement on this task related to the data representation, the need for malware specific ML techniques, and a larger training set of malware samples labeled with behaviors.

CCS CONCEPTS

• **General and reference** → **Surveys and overviews**; • **Security and privacy** → **Malware and its mitigation**; • **Computing methodologies** → **Supervised learning by classification**; Feature selection.

KEYWORDS

malware detection, supervised machine learning, benchmark dataset; behavioral labeling

ACM Reference Format:
Michael R. Smith, Nicholas T. Johnson, Joe B. Ingram, Armida J. Carbajal, Bridget I. Haus, Eva Domschot, Ramyaa Ramyaa, Christopher C. Lamb, Stephen J. Verzi, and W. Philip Kegelmeyer. 2020. Mind the Gap: On Bridging the Semantic Gap between Machine Learning and Malware Analysis. In *13th ACM Workshop on Artificial Intelligence and Security (AISec'20), November 13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3411508.3421373

1 THE PROMISE OF MACHINE LEARNING

Recently, machine learning (ML) performance has improved significantly—particularly in deep learning (DL), a class of ML algorithms that uses multiple layers in a neural network. State-of-the-art performance has been achieved in computer vision [28, 75], medical diagnosis [19], machine translation [52, 77], and game play [42, 64]. This creates hype for similar success in different applications including malware detection. ML and DL in malware detection promises to reduce manual labor by orders of magnitude, reduce errors, work at scales and speeds previously unobtainable, and detect novel malware [22, 53]. As such, many anti-virus (AV) companies are turning to ML and DL to improve malware detection and mitigation.

Despite some success of ML in MA, we observe that the results achieved in other domains have not yet been obtained in MA. We hypothesize that a semantic gap exists between the ML and malware analysis (MA) communities. MA typically identifies malware based on observed malicious or unintended behaviors requiring manual examination. It has yet to establish meaningful behavioral categories, or create realistic and challenging benchmark datasets. ML blithely uses easy benchmark datasets, focuses merely on malware classification, and is easily satisfied by its artificial success. We believe that aligning the two communities will facilitate the development of ML and data processing techniques specific for MA, and improve its performance in identifying novel malware.

Representative datasets are especially important to ML, yet data collection is problematic in MA. Many training sets are inherently biased, leading to model over-performance [13, 46] and samples with identical functionality can be mislabeled because of obfuscation techniques [29, 78]. The ML culture generally emphasizes demonstrated performance improvements on benchmark datasets [65] which has driven significant improvements but is completely dependent on an appropriate dataset. We suggest that ML-based MA can be improved by aligning the data used by ML with the goals of the MA community—specifically incorporating behavioral information. With the end goal of aligning the two communities and improving the identification of novel malware, we 1) provide ML perspectives that have led to its success in other domains that may be lacking in MA, 2) survey current datasets that are used by ML for malware detection, 3) develop a method for behavioral annotations aligned with the MITRE ATT&CK® Matrix [74], 4) annotate the Microsoft Malware Classification Challenge dataset [58] with behaviors, and 5) train ML models for behavioral identification. We find that behavioral identification is a more difficult and interesting problem for ML than generally realized. A simple image-based DL model achieves slightly over 50% accuracy, transfer learning from malware family identification achieves 57% - 84% depending on the malware family while a majority class predictor outperforms both models. The results suggest that behavioral classification can generalize to novel samples from malware families not included in training but that MA-specific ML techniques are needed.

Prior work has looked at behavioral-based features [13, 20, 85] and labels [18] for ML in MA and is discussed throughout the paper. In the following Section, we first review some preliminaries for ML and MA including the dependence of ML on the data and its ability to generalize. In Section 3 we review a use case highlighting the semantic gap between ML and MA. Sections 4 and 5 analyze the datasets and features that are used. We introduce our behavioral labeling process and present initial results in Section 6. Section 7 presents our conclusions and future work.

2 PRELIMINARIES

In this section, we provide a brief overview of ML, caveats for its success, and a brief overview of program analysis (PA) techniques used by MA teams to identify the behaviors in an executable.

2.1 Machine Learning Background

We focus on *supervised* ML that learns by example from labeled data points. We denote the inputs as X and the labels or outputs as Y. Observed variables are represented in lower-case. Therefore, the i^{th} observation of X is written as x_i which can be a vector or a scalar. Following Friedman et al. [21] to more formally describe supervised ML, let

$$Y = f(x) + \epsilon \tag{1}$$

describe the data where the noise ϵ has $E(\epsilon) = 0$ independent of X. The goal of an ML algorithm, then, is to find an approximation $\hat{f}(x)$ to $f(x)$ that preserves the predictive relationship between X and Y. The approximation $\hat{f}(X)$ is learned from a training set $\mathcal{T}$ of N observed input-output pairs (x_i, y_i), $i = 1, \ldots, N$.

An ML algorithm modifies the input-output relationship $\hat{f}(x_i)$ in response to the difference between the prediction $\hat{f}(x_i)$ and the observation y_i. Each learning algorithm has an associated set of parameters θ that can be modified to alter $\hat{f}(x)$ and many are *maximum likelihood estimators* assuming that the most likely values for θ provide the largest probability of observing Y given X.

The values of θ are found by minimizing a loss function L that measures the "goodness" of the model fit as a function of θ. For example, one loss function minimizes the residual sum-of-squares (*RSS*) or, another, the cross-entropy (*CE*) loss when Y is a vector of K possible classes. Minimizing the loss function on the training data minimizes *training error*, however, the goal is to minimize error on unobserved data points (the *test* or *generalization error*). The expected generalization error can be decomposed as:

$$\begin{aligned} Err(x) &= E[(Y - \hat{f}(x))^2] \\ &= (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma^2 \end{aligned} \tag{2}$$

which is a sum, respectively, of the bias, variance, and the irreducible error. The irreducible error (σ^2) represents the inherent noise in the data (ϵ in Equation 1)—no matter how good the model is, there will be some amount of error. The bias is the difference between the average model prediction and the actual value. High bias refers to models that focus less on the training data and possibly oversimplifies the model. High variance models, on the other hand, focus more on the training data and possibly result in overly complex models. As the complexity of a model increases, the training error tends to decrease. The performance on training data is usually not a good indicator of how the model will *generalize* or how well it will perform on new data points. Thus, a trade-off between bias and variance is needed to achieve a model that generalizes the best to test data. For more details, see Friedman et al. [21].

There is an explicit dependence between training data, $\hat{f}(x)$, and the generalization error. Gathering, cleaning, and processing data requires large amounts of effort. An observed data point x_i needs to be represented in a format that an ML model can operate on. Most ML algorithms operate on vector representations. However, many interesting problems are *not* easily represented as vectors without discarding significant amounts of information. If the representation of the data does not contain the information required for the question that is being asked (e.g. is the behavior of this executable benign or malicious?) then this falls within the irreducible error (σ^2 from Equation 2) that cannot be overcome. Additionally, an ML algorithm optimizes the loss on the labels and, therefore, the label needs to be aligned with the application. Better representations and labels of the phenomena can, thus, mitigate the irreducible error.

2.2 Overlooked Caveats of ML Successes

ML has been successfully applied in several domains that is often built on decades of previous research and understanding of a given domain. For example, the success of convolutional neural networks (CNNs) [37] builds on decades of research in signal processing and data representation. The convolution is a mathematical operator that expresses the overlap between functions and can be thought of as blending one function with another. The convolutions in CNNs, are a codification of convolutions where one function is based on the data instead of being explicitly defined. One key reason for the success of convolutions is their translational invariance which is inherently important in object recognition where an object may be anywhere in the image. An analogous operator does not yet exist for binary executable analysis.

Success of ML in other domains has also been enabled by large amounts of labeled, relevant data. Li revolutionized computer vision and object detection by providing labels for relevant images [16]. Corresponding datasets do not yet exist for malware detection. Further, ML models do not always learn the intended concepts. For example, CNNs are biased towards learning texture rather than shapes and objects [24, 70]. This can make them susceptible to adversarial attacks and brittle to noise [63].

Additionally, the data in MA is significantly different from other domains in that it lacks proximity relationships, continuity, and ordinality that are assumed by many ML algorithms. For example, pixel values of 123 and 122 are close in value and neighboring pixels have an assumed proximal relationship. Code blocks can jump to various locations in a binary and values next to each other in numerical space can have significantly different meanings. Additionally, in real-world systems, goodware significantly outnumbers malware—less than 1% of all executables were reported as malware [6]. This class imbalance has been shown to exacerbate other issues in ML algorithms [72]. The combination of these issues makes applying ML to MA difficult.

2.3 Program Analysis

Program analysis (PA) consists of several processes that are used to reason about the behavior of a computer program and are leveraged in MA. PA is ultimately interested in program optimization and correctness. We highlight a subset of areas related to extracting features that could be used as input to ML algorithms.

In PA, a distinction is made between the *syntax* and the *semantics* of a program [27]. For programs, syntax is concerned with the form of expressions that are allowed (i.e. the sequences of symbols that are accepted by a compiler or interpreter). Semantics describe the effect of executing syntactically correct expressions (behavior). The semantics of a program require a defined syntax, at least at an abstract level. Identifying syntax is much easier than semantics. As shown in Section 2.1, an ML model depends on training data. If training data does not relate to behaviors, then expecting an ML model to learn them is unreasonable. Generally, extracting syntactic features is significantly simpler than extracting semantic features.

2.3.1 Static Analysis Techniques. In static analysis, a program is analyzed in a non-runtime environment. The analysis is generally performed on a version of the source code, byte code, or application binaries. Static analysis is used frequently for optimization, such as dead code elimination, or for verification such as identifying potentially vulnerable code and run-time errors. Generally, static analysis *approximates* all possible executions of a program through abstract interpretation or data-flow analysis. One challenge for static analysis is that behavior is limited to what happens internal to the program, and the environment is not analyzed.

Several static analyses techniques capture semantic information. However, many datasets used for ML favor syntactic features that are easier to capture. For example, data flow analysis (DFA) collects information about the possible states at various points in a program [69] by constructing a control flow graph (CFG) that represents the program. Each node in the CFG represents a basic block or a sequence of consecutive instructions. Control can *only* enter at the beginning of the chunk of code in a node and leaves at the end. Directed edges in the graph represent jumps between one chunk of code to another. CFGs can capture significant semantic information. However, it is not in a form that most ML algorithms can easily digest, and there are no obvious means to transform it without significant semantic loss.

Abstract interpretation [14, 15] is a theoretical framework to formalize the approximation of computing abstract semantics. Here semantics refer to a mathematical characterization of possible behavior of a program. The most precise semantics describe accurately the actual execution of a program and are called concrete semantics. Small-step, or structural oriented, semantics [50] describe a program in terms of the behaviors of its basic operations. The behavior of a program is a current state (program point and the environment) given a starting state and series of operations. For example, consider the simple code below.

```
1: n=0
2: while n < 500 do
3:     n = n+1;
4: end
5: exit
```

Analyzing the program would yield:

$$< 1, n \Rightarrow \Omega > \rightarrow < 2, n \Rightarrow 0 > \rightarrow < 3, n \Rightarrow 0 > \rightarrow < 4, n \Rightarrow 1 > \rightarrow$$
$$< 2, n \Rightarrow 1 > \rightarrow < 3, n \Rightarrow 1 > \rightarrow < 4, n \Rightarrow 2 > \cdots < 5, n \Rightarrow 500 >$$

Operational semantics, such as small-step semantics, combine logical conclusions about program syntax in order to derive semantic meaning. Assuming the interpretation of syntax is correct, this also allows for the construction of proofs about program behavior.

Big-step, or natural, semantics [32], like small-step semantics, define basic components to describe the semantics of a program. Rather than using the basic operations like small-step, big-step analytics defines the semantics of functions. More pertinent to malware classification, both are techniques that derive semantic meaning from a program and could be looked to as inspiration for features. It is worth noting that both of these techniques limit behavior to what happens internal to a program or segment. They do not take into account the effects on the full environment as this is inherently intractable and represents a key difficulty in modeling malware for ML and MA.

Another key static analysis approach over programs is symbolic execution. Symbolic execution techniques build a mathematical representation of a program based on the input and output of various subroutines or functional blocks [9, 40]. In this representation, independent variables represent key input values. Constraint solvers, for example, can then solve for the variables, identifying what kinds of inputs are required for a particular output state [30, 59]. From a vulnerability analysis perspective, this can allow analysts to identify input that can potentially lead to system failure states, which may be exploitable. Symbolic execution techniques suffer from state explosion proportional to the size and complexity of a given program [36]. Other static analysis techniques provide disassembly and intermediate representations (from binary to machine code). However, care needs to be taken to preserve semantic information.

2.3.2 Dynamic Analysis Techniques. Dynamic analysis executes a program and *precisely* analyzes a single or limited number of executions of a program. The coverage of dynamic analysis is dependent on the test inputs, which for malware analysis, can be variants of the operating environment. Often, a subset of the interactions with the underlying operating system are analyzed such as system calls, or memory reads and writes. Dynamic analysis is often used to ensure program correctness and find errors in code [43].

Most dynamic analysis techniques use instrumentation to insert code into a program to collect run-time information. The instrumentation will vary based on the type information that is desired and the type of code that is available (e.g. source code, static binary, and dynamic binary). Most tools track function calls (including system calls), capture the input parameters, track application threads, intercept signals, and instrument a process tree. The output from dynamic analyses has often been heralded by ML practitioners for modeling behavior as it captured observed effects on the environment. However, because of a lack of context and the challenges outlined previously, the representations that are suitable for ML often lose the semantic information.

3 MOTIVATING CASE STUDIES

To help motivate the semantic gap between ML and MA , we walk through a case of using ML to identify malware persistence in registry keys—highlighting the difficulty in generating an appropriate dataset and extrapolating results to real-world scenarios.

Briefly, the registry is a hierarchical key-value database that stores configurations, program settings, and user profiles. The registry is capable of storing commands to execute when the system is loaded and is commonly used for maintaining persistence on the Windows operating system [41]. In addition to system software, malware takes advantage of the the registry to ensure that it is loaded as needed. As an example, a key can have the format:

```
\HKEY\_LOCAL\_MACHINE\System\...\...\ImagePath
```

and a value that can take multiple formats such as:

```
C:\Windows\System32\svchost.exe -k netsvcs
```

The example represents a path to an executable, but the values are capable of storing many complex data types (e.g., binary data, scripts, etc.). Thus, even with this relatively simple example, representing this data in a format suitable for ML is non-trivial.

3.1 Data Collection & Parsing

As with most use cases, collecting data is not challenging, but obtaining labels and properly representing the data is. Registry data was collected from Windows machines across a corporate network for two years, resulting in approximately 20 million (host, registry key, timestamp) tuples, with roughly 136,000 unique registry entries. Registry data was collected from executing publicly available malware in a sandbox environment producing 200 registry entries.

Despite capturing effects on the environment, the raw registry data is not suitable for ML algorithms due its variability. As there are a finite number of keys, they are represented as a 1-of-N encoding. The value portion is more complex and describes what is being executed. Ideally, the value consists of a path and a file that can be parsed into its relative components. However, in some cases one program will launch another such as when services are launched using `svchost.exe`. For these situations, a parser that found the launching program (e.g., `svchost`) as well as the program that is being launched. Each launching program is parsed according to the expected syntax (e.g., `svchost` should have a `-k` flag), and when found, these launching programs constitute another categorical variable. Additionally, different file types exist which are represented as categorical variables per file type including any associated options (e.g., command-line flags).

After the aforementioned parsing, the specific folders in a given path are used as terms in a traditional bag-of-words model. The resulting data is high-dimensional (over 12,000 terms) and extremely sparse with few unique observations (i.e., the number of unique rows is close to the number of columns). Principal Component Analysis (PCA) was performed to reduce the dimensionality while preserving as much information about the original space as possible. Several assumptions and trade-offs were made to produce a format suitable for ML which discarded some information.

3.2 Experimental Analysis and Bias

Labels are needed to identify which registry keys are associated with malicious or benign activity. Initially, any key that occured on a large number of hosts was labeled benign and those that were modified by the malware as malicious. Experimentation with this setup resulted in a cross-validated area-under-the-curve (AUC) score of 0.99. Performance this high should suggest that the ML problem is too simple and thus will not be practically useful. Upon closer inspection, the malicious examples came from specific hosts and identifying the malware labels was a simple process. Registry keys that occur on a large number of systems tend to be associated with programs and drivers in the system space (e.g., in `C:\Windows\system32`). However, the majority of the malicious keys are associated with the user and program space. A simple weak indicator that looks for absence of the keywords “windows”, “system”, or “program” to determine maliciousness provides an AUC of 0.85. Thus, the model inadvertently distinguishes system space keys versus other keys and is not likely to generalize well.

Only labeling keys modified by malware as malicious and all others as benign results in an AUC of 0.96 for ML and an AUC of 0.53 for the weak indicator—not significantly better than random. This result is promising as the gap between ML and a simple indicator increased significantly. However, this correction is likely still optimistic. Cross-validation tends to be optimistic in general, due to the fact the errors are not independent. Also, this data is not likely to contain all possible examples of malware that uses legitimate software registry for persistence. Creating a generalizing principle beyond a signature is challenging. Another confounding factor is that malware can execute behavior that is not malicious to avoid detection, and, thus, make it difficult to derive ground-truth labels.

3.3 Other Examples

This paper is not the first to recognize the gap between the research and actual deployments. Sommer and Paxson [73] point out the discrepancies in network intrusion detection. They observe that the task of intrusion detection is fundamentally different from other applications, making it more challenging. They identify six key challenges: 1) ML is better for finding similarities rather than differences, 2) very high cost of classification errors, 3) a semantic gap between detection results and their operational interpretation, 4) enormous variability in what is "normal", 5) difficulties in sound evaluation of the results, and 6) operating in an adversarial setting. In the context of detecting malware, other work noted discrepancies particularly with respect to the precision of malware—indicating a large jump in false negatives when deployed in real-world settings stemming from the difference in the proportion of malware and the difficulty on modeling "normal" in executables [71].

4 CURRENT DATASETS

In this section, we briefly discuss the importance of benchmark datasets historically in ML research and the challenges in curating a benchmark dataset for malware, and we review existing datasets. Despite several attempts, a benchmark dataset for malware classification has yet to be widely adopted and have a high impact for ML-based malware classification.

4.1 The Utility of Benchmark Datasets

The progress of any research field depends on reproducible comparisons between methods to quantify progress on a given task. For ML, benchmark datasets facilitate comparisons between learning algorithms. In addition, benchmark datasets drive ML success and guide research in several application areas such as object detection [17], facial recognition [49], handwriting recognition [38], recommender systems [26], and question and answer systems [55].

Benchmark datasets facilitate research that would not otherwise be possible. A benchmark dataset dictates several important characteristics of the research that uses it. First, it determines which features are used based on data representation. Second, it determines the impact of ML models developed using the data. If the dataset misrepresents the real-world settings or is ill-suited for the task, the ML model will perform poorly despite performing well on the benchmark dataset.

4.2 Challenges in Curating a Malware Dataset

4.2.1 Dynamic Environment. . Malware classification is a dynamic problem in which the target is constantly changing and evolving. In ML parlance, this is concept drift where the distribution of the target changes over time from what was used for training [23]. In cases with concept drift, performance often degrades and has been shown to be significant in malware detection [35]. Additionally, malware authors intentionally alter malware to avoid detection using several obfuscation techniques including polymorphic code and garbage code insertion. In many other domains, the attempt to deceive is not as prevalent. Malware authors can purposefully alter their malware to subvert an ML model trained on a given dataset.

4.2.2 Releasing Data. . Many AV companies hold their collection of malware samples as proprietary. As mentioned above, malware authors could also use this information to thwart existing architectures built on this data—risking their clients' systems. Another consideration is that each collection service may be biased to certain demographics, location, network infrastructure, political ties, etc. that may attract certain types of attacks.

4.2.3 Feature Representation. . Distributing live malware samples is a security risk, especially for those not accustomed to handling malware. As a result, most recent datasets first extract predetermined features from a set of malware examples limiting the representation.

4.2.4 Obtaining Labels. . Many of the current datasts use tools like VirusTotal [3], which provide the output from multiple antivirus tools, to create labels. Often only samples that are identified as malware by a majority of the tools are labeled as malware and others are discarded providing a biased sample that uses the most popular examples. This is not representative of the data that will be encountered in real-world deployments. Using the most popular malware and goodware examples can create easily separable training data, and overly optimistic performance expectations [39, 47]. Several works have proposed methods for improving the labeling and not discarding as many of the "unpopular" samples [34, 66].

4.3 Review of Datasets

There are currently several proposed repositories for ML-based malware detection that either identify malware families or discriminate malware from goodware; these are summarized in Table 1.

4.3.1 Live Malware Repositories. There are several repositories containing live malware—posing a threat to inadvertently infecting one's system and providing malicious software to adversaries. However, the malware samples provide a valuable resource enabling MA and research. VX (Virus eXchange) heaven [4], with the mantra: "Viruses don't harm, ignorance does!" seeks to provide information about computer viruses including articles, source code, malware samples, and books to help educate whomever is interested. Several similar repositories exist including theZoo (a.k.a. the malware DB) [1] and Virus Share [2] for free, or Virus Total [3] which is available for a fee and also contains benign samples.

Ideally, a researcher has access to the raw data. Even with access to the entire malware sample, as discussed previously, getting the samples into a format suitable for ML is challenging. Often only simple features are extracted such as metadata from the PE header, imported DLLs, and byte counts (more details on extracted features are given in Section 5). Using simple features resulted in high detection rates (98.8%) [80] leaving little room for improvement. With live malware repositories, studies are difficult to compare as each selects different subsets of malware samples to analyze and there

Table 1: Summary of malware datasets used for ML

Dataset	Year	Cite	Representations	# Samples	Labels	Labeling	Max Acc
Highly Cited							
VX Heaven [4]	2010	?	Live executables	Varies	Varies	Curated	N/A[1]
VirusShare [2]	2011	> 300	Live executables	Varies	Varies	Curated	N/A[1]
MalImg [44]	2011	417	Gray-scale images	9,458	25 Families	MSSE	99.80%
MS Malware Classification [58]	2015	76	Disassembly and hexadecimal	10,868	9 Families	MSSE	99.97%
EMBER [10]	2017	46	Parsed and histogram counts	1,100,000	Good, Bad, ?	VirusTotal	99.90%
MalRec [67]	2018	11	System calls, memory contents[2]	66,301	1,270 families	VirusTotal[3]	N/A[1]
Less Cited							
Malware Training Sets [57]	2016	2	Counts from analysis reports	4764	4 families	Curated	-
Mal-API-2019 [12]	2019	1	System call traces	7,107	8 families	VirusTotal	-
Meraz'18 Kaggle [5]	2018	~1	Parsed features	88,347	Good v Bad	Curated	91.40%[4]

[1] There is no established dataset making comparisons between studies difficult.
[2] Also provides full system replays of malware execution, however the authors note non-trivial efforts to get them to work on other systems.
[3] Uses AVClass [66] which leverages VirusTotal.
[4] Reported accuracy on the Kaggle challenge leader board.

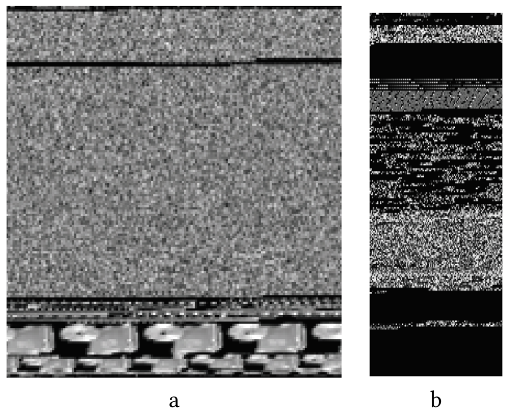

Figure 1: Examples of malware represented as gray-scale images from a) Fakerean and b) Dontovo.A malware families [44].

is no common base publication to trace attribution. However, the amount of malware samples is impressive. On Virus Share, there over 34 million samples as of this writing.

4.3.2 *MalImg.*

The MalImg dataset [44] was motivated by the success of deep learning (DL) in image processing. In MalImg, binary values from an executable were converted to 8 bit unsigned integers, organized into a 2-dimensional array and visualized as a gray-scale image (Figure 1).

The authors observed that malware belonging to the same family were visually similar in layout and texture. In their preliminary analysis, the authors extracted texture features from the generated gray-scale images using GIST [76]. On a dataset with 9,458 malware samples from 25 different families, a 3-nearest neighbor classifier[1] achieved 97.18% accuracy and 99.2% when variants of a malware family were combined. Follow-up work achieved accuracy of 98.52% when using a convolutional neural network [33] and 99.80% with principal component analysis and a support vector machine [25].

[1]A classifier that predicts the majority class of the 3-closest examples

4.3.3 *MS Malware Classification.*

The Microsoft Malware Classification Challenge [58] was developed as a Kaggle competition to classify malware samples into one of nine malware families. It was released in 2015 and has since been used in several studies, being cited more than 70 times at the time of this writing. The hexadecimal representation of the binary content without the PE header as well as meta-information (function calls, op codes, strings, etc.) from the IDA disassembler was provided for each malware sample. Current reported performance on the dataset claims 99.70% [25] and 99.97% accuracy [33] using image-based features.

4.3.4 *EMBER.*

The Endgame Malware BEnchmark for Research (EMBER) dataset [10] is a collection of extracted features from 1.1 million executables divided into 900k training and 200k test samples and has emerged as one of the most popular datasets. EMBER provides features that are consistent with previous work and has been used in several studies. The authors of EMBER achieved a 98.2% detection rate with a 1% false positive rate. This was further improved to a 99.4% detection rate with an AUC value of 0.9997 [48]. Further, Vinayakumar et al. [79] modify a DL technique aimed at malware detection (MalConv [53]) and achieve nearly perfect performance.

4.3.5 *Malrec.*

Contrary to the other datasets, Malrec provides system-wide traces of malware executions that can be replayed. It is intended to address the danger of releasing live malware and the limited amount of data that can be collected when running in a sandbox. The replays capture the state of a system that is executing malware and thus captures the behaviors of malware while not releasing actual malware and provides the ability to retrospectively extract features that were not considered relevant when the

Table 2: Reported accuracy, precision, recall and F1-score on the EMBER dataset [79].

Model	Accuracy	Precision	Recall	F1
MalConv [53]	98.8%	99.7	97.9	98.8
GBDT [10]	97.5%	99.0	96.2	97.1
KNN	95.1%	95.5	94.6	95.1
DT	96.9%	97.1	96.7	96.9
RF	97.0%	98.6	95.3	96.9
SVM	96.1%	96.4	95.7	96.1
DNN	98.9%	99.7	98.1	98.9
Modified MalConv [79]	99.9%	99.7	100.0	99.9

malware was first executed. There are currently 66,301 malware recordings collected over a two-year period. The major downside is the very large size of the data (currently 1.3TB) and the complexity in setting up the system to extract a dataset.

The authors extracted multiple datasets from the system-wide recordings including bag-of-word counts for textual data in memory, network activity, system call traces, and counts of data instruction mnemonics. They examined a use case in which they extracted features to use for ML. They created a word list of all words between 4 and 20 characters long from the English Wikipedia—resulting 4.4 million terms. They then monitored memory reads and writes looking for byte sequences that matched words in their list. Terms were removed that appeared in a baseline of running goodware as well as frequent terms that appeared in more than 50% of the samples and rare terms that appeared in less than 0.1% of the samples resulting in ~460,000 terms. The dimensionality was further reduced to 2048 input features using PCA. DL on this data achieved a median F1-score of 97.2% across all of the malware families.

Despite having system-wide information, the PCA summary was sufficient for their dataset to achieve high accuracy. This presents somewhat of a paradox in the claims of ML and what is observed in deployed systems. Analyzing the memory contents in a bag-of-words fashion loses context, and we argue, that it is akin to learning a signature. We conclude that the ML model is able to quickly learn an effective signature-based malware detection system.

4.3.6 *Other Datasets.*

Other datasets have been created, often by other security companies and hobbyists [5, 12, 57]. These datasets have not been widely adopted nor is it apparent how much maintenance they receive. We include them here for completeness, but they do not provide any new feature representations.

4.3.7 *ML Perspectives.*

From an ML perspective, achieving such high classification accuracy is somewhat concerning as there is fear that the model has either overfit the training data (will have high generalization error) or the training data is easily separable. Thus, the dataset may not represent real world conditions well and give unrealistic performance expectations. For EMBER, the authors point out that the classes were easy to correctly classify and have attempted to make the task more challenging [61] in addition to other modifications [60]. The baseline on the updated data is 86.8%. Unfortunately, there are few results on the updated dataset.

5 ANALYSIS OF DATASETS AND FEATURES

In this section, we examine which features contribute to the performance of an ML model across the datasets. We find that 1) the most useful features vary across datasets and 2) very few papers attempt to extract semantic features *and* are careful to maintain semantic information. We suggest that the ML models operate on patterns in the data not used by the MA community and that syntactic features are useful for discriminating between existing malware classes similar to how non-intuitive textures in images are useful for object detection. Similar to object detection in images, the models should not be expected to detect novel forms of malware based on their behavior as there is a semantic gap between the data and the task.

Raman [56] examined which features are the most discriminative between malware samples from VX Heaven and software that comes installed by default on Windows operating systems. They were able to achieve a true positive rate of 98.6% with a false positive rate of 5.7% by only extracting *seven* features from the files. Further examination revealed that the ML algorithm learned to discriminate between Microsoft and non-Microsoft executables. As the dataset does not represent the real-world problem well, it affects the robustness of an ML model trained on that data. A high false negative rate would be expected with a broader set of goodware.

Ahmadi et al. [8] extracted a large number of features that are commonly used in ML models from the hexadecimal representation and disassembled files from the Microsoft Malware Classification Challenge dataset with the intent of identifying features that are the most discriminative. The examined features include:

(1) byte counts (BYTE).
(2) the size of the hexadecimal representation and the address of the first byte sequence (MD1).
(3) byte entropy (ENT).
(4) image representation using Haralick features (IMG1) and Local Binary Patterns (IMG2).
(5) histogram of the length of strings extracted from the hexadecimal file (STR).
(6) the size of, number of line in the disassembled file (MD2).
(7) the frequency of a set of symbols in the disassembled file (-, +, *,], [, ?, @) (SYM).
(8) the frequency of the occurrence of a subset of 93 of possible operation codes in the disassembled file (OPC).
(9) the frequency of the use of registers (REG).
(10) the frequency of the use of the top 794 Window API calls from a previous analysis of malware (API).
(11) characteristics of the sections in the binary (SEC).
(12) statistics around using db, dw, and dd instructions which are used for setting byte, word, and double word and are used to obfuscate API calls (DP).
(13) the frequency of 95 manually chosen keywords from the disassembled code (MISC)

Table 3 shows the classification accuracy on the training set and from using 5-fold cross-validation for each subset of extracted features using gradient boosted decision trees. There are several feature groups that achieve over 99% accuracy including MISC which counts the occurrence of a set of hand-selected keywords. Surprisingly, MD1 and MD2 (i.e. file size) achieve about 85% and 76% accuracy respectively (random is 11.11%). This highlights a concern

Table 3: The reported accuracy on the training set and using 5-fold cross-validation on the Microsoft Malware Classification Challenge dataset [8].

Feature	# Features	Train Accuracy	5-CV Accuracy
Hexadecimal file			
ENT	203	99.87%	98.62%
BYTE	256	99.48%	98.08%
STR	116	98.77%	97.35%
IMG1	52	97.18%	95.50%
IMG2	108	97.36%	95.10%
MD1	2	**85.47%**	**85.25%**
Disassembled file			
MISC	95	99.84%	99.17%
OPC	93	99.73%	99.07%
SEC	25	99.48%	98.99%
REG	26	99.32%	98.33%
DP	24	99.05%	98.11%
API	796	99.05%	98.43%
SYM	8	98.15%	96.84%
MD2	2	**76.55%**	**75.62%**

that there are features which may be discriminative but are an artifact of the dataset and can easily be manipulated adversarially.

Oyama et al. [45] examine which features have the largest impact on the EMBER dataset. EMBER contains several feature groups:

(1) General file information from the PE header such as virtual size of the file, thread local storage, resources, as well as the file size and number of symbols.
(2) Header information from the COFF header providing the timestamp, the target machine, linker versions, and major and minor image versions.
(3) Import functions obtained by parsing the address table.
(4) Exported functions.
(5) Section information including the name, size, entropy virtual size and list of strings representing section characteristics.
(6) Byte histogram representing the counts of each byte value.
(7) Byte-entropy histogram approximating the joint distribution of entropy and a given byte value.
(8) Simple statistics about printable strings that are at least five characters long. Specifically providing information on strings that begin with “C:\", “http://", “https://" or “HKEY_".

Table 4 shows the accuracy for each feature group. The imports, which also have the largest number of features, has the highest accuracy as 77.8%. Oyama et al. report that header, imports, section, and histogram feature groups (together) achieve about 90% accuracy. The remaining 2.7% comes from the other feature groups.

Other work makes similar observations on various datasets further indicating a needed change in data representation:

- Count features (histograms) promotes overfitting and, combined with the labels, produces overly optimistic results [54].
- PE headers are the most discriminative [84].

Table 4: Reported accuracy and number of features for each feature set in the EMBER dataset [45].

Feature set	Number of features	Accuracy
imports	1280	77.8
section	255	68.2
histogram	256	68.1
byte entropy	256	61.8
strings	104	61.4
general	10	56.0
header	62	52.9
exports	128	17.2
All	2,351	92.7

- On VX Heaven, PE-Miner [68] achieves a detection rate greater than 99% only using structural information (PE and section header information), DLLs and object files.

Despite the impressive results, none of the features capture behaviors as the data is not tailored to provide that information and the ML task is to detect malware—*not* identify behaviors.

6 BEHAVIORAL-BASED DATASETS

As we have shown, most datasets have focused on the features which do not contain behavioral information or it is lost when extracting features. As a first step to modeling behaviors, we take an alternative approach and provide labels for the behavior expressed in malware so the ML model can search for behavioral artifacts. Extracting behavioral information from an executable is a challenging problem that is a current research area for MA. We offer a process using threat reports for malware families to gather behavioral information.

We propose that behaviors consist of 1) a high-level intent and 2) low-level “primitives” that accomplish the behavior. A primitive is a sequence of ordered or partially ordered (i.e., one step depends on the previous step(s)) steps that must occur for the behavior to be successful. It is possible that primitives may contain conditional statements that are represented better by a directed graph than a sequence. These primitives vary by representation, malware family, or malware toolkit. They may also involve multiple systems (e.g., network and host). Thus, the feature representation is non-trivial. Additionally, the high-level intent of the executable is often not in the data. Further, multiple primitives may exist that accomplish the same behavior. For example, persistence is a common behavior for malware. This same outcome can be achieved variously by copying the malware to the *startup* folder or modifying the registry.

To label the behaviors, we leverage the MITRE Malware Behavior Catalog (MBC) [7]. MBC supports MA mapping behavior onto the MITRE ATT&CK Matrix [74]. ATT&CK documents common tactics, techniques, and procedures that advanced persistent threats use against Windows enterprise networks. The behaviors are organized according to the **objective** of the malware such as *Anti-Behavioral Analysis*, *Command and Control*, or *Persistence*. Each objective contains behaviors and code characteristics (**techniques**) that support

Table 5: Malware Behavior Label Example for Microsoft Malware Classification Challenge

Objective:	Collection		Credential Access				Defense Evasion			...
Technique:	Local System	Man in the Browser	Hooking	Steal Web Session	Credential in Web Browser	Credentials in Files	Masquerading	Disable Sec Tools	Process Injection	...
Gatak	x	-	x	-	-	-	x	-	x	...
Ramnit	x	x	x	x	x	-	-	x	x	...
Lollipop	x	-	-	-	-	-	-	-	-	...
Kelihos	x	-	-	-	-	-	-	-	-	...
Vundo	x	-	-	-	-	x	x	x	x	...
Simda	x	-	-	-	-	-	x	x	-	...
Tracur	-	-	-	-	-	-	-	-	-	...

that objective. For *Persistence* some of the techniques include *Application Shimming*, *DLL Search Order Hijacking*, and *Scheduled Task*. Each technique has an explanation for what it covers and can belong to multiple objectives—the *Hidden Files and Directories* technique could be under the *Defense Evasion* or *Persistence* objective.

We label the behaviors of a malware family using open-source threat reports and map the reported behaviors to the objectives and techniques outlined by MBC. In some cases, judgment has to be made about which category is the most appropriate. We label each family multiple times and use a peer review style to come to conclusions. The behavioral labels for each family are then extrapolated to individual examples. The current process is subjective and time intensive, and errors can be made based on variations of a malware family. Despite these limitations, the behavioral labeling helps align the data to the desired task of identifying novel malware samples based on its behaviors. The labels would allow an ML model to directly learn the behaviors that may be not be discernible using only the family name. Future work includes the use of natural language processing tools to help automate the process. As new malware is analyzed, behaviors could be mapped into the MBC directly, bypassing the need for this method.

We label the Microsoft Malware Classification Challenge dataset which includes seven malware families, (*Ramnit*, *Lollipop*, *Kelihos*, *Vundo*, *Simda*, *Tracur*, *Gatak*).[2] The result of this process is a hierarchical behavioral labeling of each malware family as shown in Table 5. The compiled version is accessible at https://gitlab.com/malgen/behavior_labels/. The hierarchical structure captures both the high-level objective and employed technique(s) to meet that objective. By providing this labeling, an ML model will learn features that are associated with behaviors across all included malware families. By adjusting the target of the ML algorithms, better features and models can be developed that will improve the deployment of ML-base malware detectors.

We are not the first to suggest the addition of behavioral labels; however, our process provides richer behavioral annotation at the cost of manual process and, as shown below, is a more challenging problem. Semantic Malware Attribute Relevance Tagging (SMART) [18] uses the output from anti-virus suites and parses keywords from the output providing a richer potential set of technical feature information than other approaches [85]. For example, the output could be `Win32.Virlock.Gen.8` or `TR/Crypt.ZPACK.Gen` and the key words extracted are `Virlock`, and `Crypt` and `ZPACK` respectively. This provides information that the malware is respectively ransomware and packed. The keywords align with the objectives in our process but do not provide consistent information on how the behavior is implemented, which our method provides. They report an accuracy of 95% on 11 possible tags. Our motives are similar to those if SMART, but the finer grained labeling that our method provides facilitates improved analysis and forces a ML algorithm to learn to distinguish behaviors that are important to MA.

High-level additional information was shown to improve the performance of an ML model [62]. We anticipate similar improved results as well as adjustments in follow-on studies that focus on behaviors.

6.1 Experiments on Behavioral Labels

We examine the ability of ML to generalize to novel malware on two initial behavior classifiers trained on the behavioral annotations using a binary image representation of the malware. As a malware family can have a combination of behaviors, we treat the problem as a multi-label classification problem and use a binary cross-entropy loss to encode this multi-label objective. Behavioral labels are consistent across malware families, allowing the identification of behavior in novel malware samples. In the following experiments, we hold out one family, train the models on the remaining malware families, and calculate the average accuracy for all of the behaviors on the held-out class. This allowed us to test the model's ability to reason about behaviors for a malware family it hadn't seen before. We compare the performance of the ML models with a simple majority class predictor that predicts a behavior is present if it was present in the majority of the training samples.

We establish a baseline model with a simple convolutional architecture, used for image processing, based on [51]. Using the input size of that architecture, we selected the first 1024 bytes from our malware samples as a (32,32) black and white image. This is a limited snapshot of malware but we saw accuracies above random chance when evaluating the model on family classification.

To develop a more robust model for predicting behaviors from malware binaries, we used the MalConv architecture [53], leveraging the code and model pre-trained on the EMBER dataset described in [10]. For consistency with the baseline model, we used the first

[2] *Kelihos* versions 1 and 3 were combined because the threat reports did not distinguish between versions and we dropped Obfuscator.ACY as it was a bucket for obfuscated malware for which the family could not be determined.

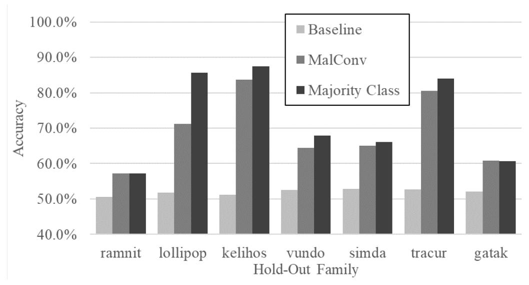

Figure 2: Classifying Behaviors for Unseen Families

megabyte of the malware sample represented as a flattened malware binary image as input. Additionally, we replaced the final fully connected layer to provide outputs for each of our behaviors, changing the output size to 56. This allowed us to fine-tune the model to evaluate the ability of transfer learning to classify malware behaviors based on features extracted for malware detection.

In Figure 2, we present the average accuracy across all behaviors for each variant of the experiment. Our transfer learning approach (MalConv)[3] outperforms our baseline model but the Majority Class classifier achieves better performance than both of them. This highlights the challenge ML faces when classifying behaviors for MA and the work that still needs to be performed. Since all of our test samples are from the same family, (i.e. labeled with the same behaviors), the perfect classifier would predict the same labels for each sample. If we had more families and could hold out multiple families for evaluating generalizability, then a naive classifier might not be as successful. Further, we examined the correlation of extracted features with the behavioral annotations. We used the features that were used by the winning team of the 2015 Kaggle Microsoft Malware Classification Challenge [81] (7600 features in total). Pearson correlations were calculated between the extracted features and the behaviors. Only 70 of the features had correlation $r > 0.7$ with any behavior. 64 of the 70 features that are strongly correlated with a behavior are 4-gram byte counts. Raff et al. concluded that n-byte grams were most often picking up string features [54]. One of the behaviors with a strong correlation with byte-grams is "rootkit", which often inserts malicious code into commonly used processes such as DLLs which are often in disassembled string information and are one type of information in the 4-gram byte counts that could be correlated with behaviors. The behavior "access credentials" has a strong correlation with the instruction used to shift the bits in the register or memory, and is often used to encode or decode information.

The initial results presented here are not fully optimized but highlight a semantic gap between ML and MA based on the data used for analyses. There are several aspects that could be examined including balancing the behaviors, data generation, and hyper-parameter tuning. The results suggest that generalized behavior classification may be a more difficult problem than classifying malware families. They highlight the need for a dataset with behavioral labels and that simply using techniques that work well in other domains *does not* directly transfer to behavioral identification.

[3]See Al Kadri et al. [31] for a more focused approach of applying transfer learning to MalConv for predicting malware families.

7 CONCLUSIONS

In this paper, we reviewed the body of research on providing datasets to train ML models for the classification of malware. We suggest that current feature extraction and current ML techniques optimized for signal processing are inadequate for malware behavior detection. As ML is being used by an increasing number of AV companies, it is important that the lessons learned from the successful development of ML in other areas is also used in MA. This is accomplished through building on a strong foundation of the application domain. We believe that bridging the ML and MA communities will align the questions that are being asked to how they are answered. We have shown that this misalignment in the ML domain stems from a semantic gap in the available data and how that data is represented. While both communities seek to identify malware, the MA community uses semantic parsing techniques to try to understand what the program is doing and, based on the behavior, a decision can be made to determine if the program is malware or goodware. The ML community has primarily focused on determining the intent of a program to classify malware. For ML to make a larger impact in deployed settings, we advocate for 1) an increased collaboration between the two communities, 2) more behavior-based features in data sets and the inclusion of samples that are not clear cut goodware or malware, and the inclusion of behavioral information, 3) modifying the task of determining intent to identifying behaviors, and 4) the development of a benchmark dataset that more closely aligns with problems encountered by the MA community. Benchmark datasets have a history of driving significant improvements in ML in a given application area (i.e. computer vision) and an appropriate one could help drive the ML-based MA. As a first step, we proposed a method for annotating datasets with behavioral information and annotated the Microsoft Microsoft Malware Classification Challenge dataset.

As new ML and DL methods are developed, some may have more applicability outside of simply classifying malware. Attention [11, 82] was introduced as a method to help a DL method focus on the most pertinent portions of the input. In the MA community, often a piece of software needs to be partially reverse engineered to understand the behavior of the software. Attention allows for an ML model to learn which portions of an executable are the most pertinent to resulting classification [83]. Augmented with the behavioral annotations, attention would also indicate which portions of an executable are the most pertinent to that behavior. This would result in significant decreases in analyst time and potentially lead to improved program understanding.

ACKNOWLEDGMENTS

This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND2020-6788 C

REFERENCES

[1] [n.d.]. theZoo - A live malware repository. https://thezoo.morirt.com/ (Accessed July 2020).

[2] [n.d.]. VirusShare.com - Because Sharing is Caring. https://virusshare.com/ (Accessed June 2020).

[3] [n.d.]. VirusTotal. https://www.virustotal.com (Accessed July 2020).

[4] [n.d.]. VX Heaven Virus Collection. http://vxheaven.org/ (Accessed July 2020).

[5] 2018. Malware detection – make your own malware security system, in association with meraz'18 malware security partner Max Secure Software. https://www.kaggle.com/c/malware-detection.

[6] 2019. *2019 WEBROOT Threat Report.* Technical Report. Webroot, Broomfield, Colorado.

[7] 2019 (Accessed July 2020). Malware Behavior Catalog. https://github.com/MBCProject/mbc-markdown

[8] Mansour Ahmadi, Dmitry Ulyanov, Stanislav Semenov, Mikhail Trofimov, and Giorgio Giacinto. 2016. Novel Feature Extraction, Selection and Fusion for Effective Malware Family Classification. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy* (New Orleans, Louisiana, USA) *(CODASPY '16)*. Association for Computing Machinery, New York, NY, USA, 183–194.

[9] Saswat Anand, Patrice Godefroid, and Nikolai Tillmann. 2008. Demand-driven compositional symbolic execution. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 367–381.

[10] Hyrum S. Anderson and Phil Roth. 2018. EMBER: An Open Dataset for Training Static PE Malware Machine Learning Models. *CoRR* abs/1804.04637 (2018).

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.).

[12] Ferhat Özgür Çatak and Ahmet Faruk Yazi. 2019. A Benchmark API Call Dataset for Windows PE Malware Classification. *CoRR* abs/1905.01999 (2019).

[13] Lorenzo Cavallaro. 2019. When the Magic Wears Off: Flaws in ML for Security Evaluations (and What to Do about It). (2019).

[14] Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Los Angeles, California). Association for Computing Machinery, New York, NY, USA, 238–252.

[15] Patrick Cousot and Radhia Cousot. 2014. Abstract Interpretation: Past, Present and Future. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (Vienna, Austria). Association for Computing Machinery, New York, NY, USA, Article 2, 10 pages.

[16] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 248–255.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *IEEE Computer Vision and Pattern Recognition (CVPR)*.

[18] Felipe N. Ducau, Ethan M. Rudd, Tad M. Heppner, Alex Long, and Konstantin Berlin. 2019. SMART: Semantic Malware Attribute Relevance Tagging. *CoRR* abs/1905.06262 (2019). arXiv:1905.06262 http://arxiv.org/abs/1905.06262

[19] Bradley J. Erickson, Panagiotis Korfiatis, Zeynettin Akkus, and Timothy L. Kline. 2017. Machine Learning for Medical Imaging. *RadioGraphics* (Feb 2017). https://doi.org/10.1148/rg.2017160130

[20] I. Firdausi, C. lim, A. Erwin, and A. S. Nugroho. 2010. Analysis of Machine learning Techniques Used in Behavior-Based Malware Detection. In *2010 Second International Conference on Advances in Computing, Control, and Telecommunication Technologies*. 201–203.

[21] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning*. Vol. 1. Springer series in statistics New York.

[22] Hisham Galal. 2015. Behavior-based features model for malware detection. *Journal of Computer Virology and Hacking Techniques* (06 2015). https://doi.org/10.1007/s11416-015-0244-0

[23] João Gama, Indrundefined Žliobaitundefined, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. 2014. A Survey on Concept Drift Adaptation. *ACM Comput. Surv.* 46, 4, Article 44 (March 2014), 37 pages. https://doi.org/10.1145/2523813

[24] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. 2019. ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[25] Lahouari Ghouti. 2020. Malware Classification Using Compact Image Features and Multiclass Support Vector Machines. *IET Information Security* (01 2020). https://doi.org/10.1049/iet-ifs.2019.0189

[26] F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems* 5, 4, Article 19 (Dec. 2015), 19 pages.

[27] Matthew Hennessy. 1990. *The Semantics of Programming Languages: An Elementary Introduction Using Structural Operational Semantics*. John Wiley & Sons, Inc., USA.

[28] J. Hu, L. Shen, and G. Sun. 2018. Squeeze-and-Excitation Networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 7132–7141.

[29] Grégoire Jacob, Paolo Milani Comparetti, Matthias Neugschwandtner, Christopher Kruegel, and Giovanni Vigna. 2012. A static, packer-agnostic filter to detect similar malware samples. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 102–122.

[30] Joxan Jaffar and J-L Lassez. 1987. Constraint logic programming. In *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. 111–119.

[31] Mohamad Al Kadri, Mohamed Nassar, and Haidar Safa. 2019. Transfer learning for malware multi-classification. In *Proceedings of the 23rd International Database Applications & Engineering Symposium*. 1–7.

[32] Gilles Kahn. 1987. Natural Semantics. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science (STACS '87)*. Springer-Verlag, Berlin, Heidelberg, 22–39.

[33] M. Kalash, M. Rochan, N. Mohammed, N. D. B. Bruce, Y. Wang, and F. Iqbal. 2018. Malware Classification with Deep Convolutional Neural Networks. In *2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*. 1–5.

[34] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. D. Tygar. 2015. Better Malware Ground Truth: Techniques for Weighting Anti-Virus Vendor Labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security* (Denver, Colorado, USA). Association for Computing Machinery, New York, NY, USA, 45–56.

[35] W. P. Kegelmeyer, K. Chiang, and J. Ingram. 2013. Streaming Malware Classification in the Presence of Concept Drift and Class Imbalance. In *2013 12th International Conference on Machine Learning and Applications*, Vol. 2. 48–53.

[36] Volodymyr Kuznetsov, Johannes Kinder, Stefan Bucur, and George Candea. 2012. Efficient state merging in symbolic execution. *Acm Sigplan Notices* 47, 6 (2012), 193–204.

[37] S. Lawrence, C. L. Giles, Ah Chung Tsoi, and A. D. Back. 1997. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks* 8, 1 (1997), 98–113.

[38] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. In *Proceedings of the IEEE*, Vol. 86. 2278–2324.

[39] Peng Li, Limin Liu, Debin Gao, and Michael K. Reiter. 2010. On Challenges in Evaluating Malware Clustering. In *Recent Advances in Intrusion Detection*, Somesh Jha, Robin Sommer, and Christian Kreibich (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 238–255.

[40] Kin-Keung Ma, Khoo Yit Phang, Jeffrey S Foster, and Michael Hicks. 2011. Directed symbolic execution. In *International Static Analysis Symposium*. Springer, 95–111.

[41] Microsoft Corporation. 2018, (Accessed May 2020). Structure of the Registry. https://docs.microsoft.com/en-us/windows/win32/sysinfo/structure-of-the-registry

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (Feb. 2015), 529–533.

[43] Glenford J. Myers, Corey Sandler, and Tom Badgett. 2011. *The Art of Software Testing* (3rd ed.). Wiley Publishing.

[44] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath. 2011. Malware Images: Visualization and Automatic Classification. In *Proceedings of the 8th International Symposium on Visualization for Cyber Security* (Pittsburgh, Pennsylvania, USA) *(VizSec '11)*. Association for Computing Machinery, New York, NY, USA, Article 4, 7 pages. https://doi.org/10.1145/2016904.2016908

[45] Y. Oyama, T. Miyashita, and H. Kokubo. 2019. Identifying Useful Features for Malware Detection in the Ember Dataset. In *2019 Seventh International Symposium on Computing and Networking Workshops (CANDARW)*. 360–366.

[46] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium (USENIX Security 19)*. 729–746.

[47] Roberto Perdisci and ManChon U. 2012. VAMO: Towards a Fully Automated Malware Clustering Validity Analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference* (Orlando, Florida, USA) *(ACSAC 2012)*. Association for Computing Machinery, New York, NY, USA, 329–338.

[48] Huu-Danh Pham, Tuan Dinh Le, and Thanh Nguyen Vu. 2018. Static PE Malware Detection Using Gradient Boosting Decision Trees Algorithm. In *Future Data and Security Engineering*, Tran Khanh Dang, Josef Küng, Roland Wagner, Nam Thoai, and Makoto Takizawa (Eds.). Springer International Publishing, Cham,

228–236.

[49] P. Jonathon Phillips, Harry Wechsler, Jeffrey Huang, and Patrick J. Rauss. 1998. The FERET database and evaluation procedure for face-recognition algorithms. *Image and Vision Computing* 16, 5 (1998), 295–306.

[50] Gordon D. Plotkin. 1981. *A Structural Approach to Operational Semantics*. Technical Report DAIMI FN-19.

[51] PyTorch. 2020 (Accessed June 2020). *Training a Classifier*. https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

[52] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. (2019).

[53] Edward Raff, Jon Barker, Jared Sylvester, Robert Brandon, Bryan Catanzaro, and Charles Nicholas. 2017. Malware detection by eating a whole exe. *arXiv preprint arXiv:1710.09435* (2017).

[54] Edward Raff, Richard Zak, Russell Cox, Jared Sylvester, Paul Yacci, Rebecca Ward, Anna Tracy, Mark McLean, and Charles Nicholas. 2018. An investigation of byte n-gram features for malware classification. *Journal of Computer Virology and Hacking Techniques* 14 (2018), 1–20. https://doi.org/10.1007/s11416-016-0283-1

[55] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

[56] Karthik Raman. 2012. Selecting Features to Classify Malware. In *Proceedings of InfoSec Southwest*.

[57] Marco Ramilli. 2016 (Accessed February 2020). Malware Training Sets: a machine learning dataset for everyone. https://marcoramilli.com/2016/12/16/malware-training-sets-a-machine-learning-dataset-for-everyone/

[58] Royi Ronen, Marian Radu, Corina Feuerstein, Elad Yom-Tov, and Mansour Ahmadi. 2018. Microsoft Malware Classification Challenge. *CoRR* abs/1802.10135 (2018).

[59] Francesca Rossi, Peter Van Beek, and Toby Walsh. 2006. *Handbook of constraint programming*. Elsevier.

[60] Phil Roth. 2019 (Accessed April 2020). EMBER Improvements. https://www.camlis.org/2019/talks/roth Part of CAMLIS.

[61] Phil Roth, Hyrum Anderson, and Sven Cattell. 2019. Extending EMBER. https://www.elastic.co/blog/extending-ember [Accessed April 2020].

[62] Ethan M. Rudd, Felipe N. Ducau, Cody Wild, Konstantin Berlin, and Richard E. Harang. 2019. ALOHA: Auxiliary Loss Optimization for Hypothesis Augmentation. *CoRR* abs/1903.05700 (2019). http://arxiv.org/abs/1903.05700

[63] Lukas Schott, Jonas Rauber, Matthias Bethge, and Wieland Brendel. 2018. Towards the first adversarially robust neural network model on MNIST. *CoRR* abs/1805.09190 (2018).

[64] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. 2019. Mastering atari, go, chess and shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265* (2019).

[65] David Sculley, Jasper Snoek, Alexander B. Wiltschko, and Ali Rahimi. 2018. Winner's Curse? On Pace, Progress, and Empirical Rigor. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net.

[66] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A Tool for Massive Malware Labeling. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses (Lecture Notes in Computer Science)*, Fabian Monrose, Marc Dacier, Gregory Blanc, and Joaquín García-Alfaro (Eds.), Vol. 9854. Springer, 230–253.

[67] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. 2018. Malrec: Compact full-trace malware recording for retrospective deep analysis. In *Detection of Intrusions and Malware, and Vulnerability Assessment - 15th International Conference, DIMVA 2018, Proceedings (Lecture Notes in Computer Science)*. Springer-Verlag, 3–23.

[68] M. Zubair Shafiq, S. Momina Tabish, Fauzan Mirza, and Muddassar Farooq. 2009. PE-Miner: Mining Structural Information to Detect Malicious Executables in Realtime. In *Recent Advances in Intrusion Detection*, Engin Kirda, Somesh Jha, and Davide Balzarotti (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 121–141.

[69] Micha Sharir, Amir Pnueli, et al. 1978. *Two approaches to interprocedural data flow analysis*. New York University. Courant Institute of Mathematical Sciences ….

[70] Samarth Sinha, Animesh Garg, and Hugo Larochelle. 2020. Curriculum By Texture. arXiv:2003.01367 [cs.LG]

[71] Michael R. Smith, Joe Ingram, Christopher Lamb, Timothy Draelos, Justin Doak, James Aimone, and Conrad James. 2018. Dynamic Analysis of Executables to Detect and Characterize Malware. In *2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 16–22.

[72] Michael R. Smith, Tony Martinez, and Christophe Giraud-Carrier. 2014. An Instance Level Analysis of Data Complexity. *Machine Learning* 95, 2 (May 2014), 225–256.

[73] R. Sommer and V. Paxson. 2010. Outside the Closed World: On Using Machine Learning for Network Intrusion Detection. In *2010 IEEE Symposium on Security and Privacy*. 305–316.

[74] Blake E. Strom, Andy Applebaum, Doug P. Miller, Kathryn C. Nickels, Adam G. Pennington, and Cody B. Thomas. 2018. *MITRE ATT&CK™: Design and Philosophy*. Technical Report 18-0944-11. MITRE Corporation, McClean, VA. 37 pages.

[75] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. 2017. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence* (San Francisco, California, USA) *(AAAI'17)*. AAAI Press, 4278–4284.

[76] Antonio Torralba, Kevin P. Murphy, William T. Freeman, and MArk A. Rubin. 2003. Context-based vision system for place and object recognition. In *Proceedings Ninth IEEE International Conference on Computer Vision*, Vol. 1. 273–280.

[77] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 5998–6008. http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[78] Giovanni Vigna and Davide Balzarotti. 2018. When malware is packin' heat. In *Enigma 2018 (Enigma 2018)*.

[79] R. Vinayakumar, M. Alazab, K. P. Soman, P. Poornachandran, and S. Venkatraman. 2019. Robust Intelligent Malware Detection Using Deep Learning. *IEEE Access* 7 (2019), 46717–46738.

[80] R. Vyas, X. Luo, N. McFarland, and C. Justice. 2017. Investigation of malicious portable executable file detection on the network using supervised learning techniques. In *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*. 941–946.

[81] Xiaozhou Wang, Jiwei Liu, and Xueer Chen. 2015 (Accessed July 2020). *Microsoft Malware Classification Challenge (BIG 2015): First Place Team: Say No to Overfitting*. https://github.com/xiaozhouwang/kaggle_Microsoft_Malware

[82] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. In *Proceedings of the 32nd International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Francis Bach and David Blei (Eds.), Vol. 37. PMLR, Lille, France, 2048–2057.

[83] Hiromu Yakura, Shinnosuke Shinozaki, Reon Nishimura, Yoshihiro Oyama, and Jun Sakuma. 2019. Neural Malware Analysis with Attention Mechanism. *Computers & Security* (08 2019), 101592. https://doi.org/10.1016/j.cose.2019.101592

[84] Guanhua Yan, Nathan Brown, and Deguang Kong. 2013. Exploring Discriminatory Features for Automated Malware Classification. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Konrad Rieck, Patrick Stewin, and Jean-Pierre Seifert (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 41–61.

[85] Ziyun Zhu and Tudor Dumitraş. 2016. Featuresmith: Automatically engineering features for malware detection by mining the security literature. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 767–778.