Experimental Study of Machine Learning based Malware Detection Systems' Practical Utility

Yuping Li[1], Doina Caragea[2], Lawrence Hall[3], and Xinming Ou[3]

[1]Pinterest Inc., [2]Kansas State University, [3]University of South Florida

***Abstract*—Thanks to the numerous machine learning based malware detection (MLMD) research in recent years and the readily available online malware scanning system (*e.g.*, VirusTotal), it becomes relatively easy to build a seemingly successful MLMD system using the following standard procedure: first prepare a set of ground truth data by checking with VirusTotal, then extract features from training dataset and build a machine learning detection model, and finally evaluate the model with a disjoint testing dataset. We argue that such evaluation methods do not expose the real utility of ML based malware detection in practice since the ML model is both built and tested on malware that are known at the time of training. The user could simply run them through VirusTotal just as how the researchers obtained the ground truth, instead of using the more sophisticated ML approach. However, ML based malware detection has the potential of identifying malware that has not been known at the time of training, which is the real value ML brings to this problem. We present experimentation study on how well a machine learning based malware detection system can achieve this. Our experiments showed that MLMD can consistently generate previously unknown malware knowledge, *e.g.*, malware that is not detectable by existing malware detection systems at MLMD's training time. Our research illustrates an ideal usage scenario for MLMD systems and demonstrates that such systems can benefit malware detection in practice. For example, by utilizing the new signals provided by the MLMD system and the detection capability of existing malware detection systems, we can more quickly uncover new malware variants or families.**

I. Introduction

The total number of independent Android malware detection products listed on VirusTotal [1] increased from 54 in 2015 to 71 in 2019. A significant number of the newly added malware detection systems are known to be built with machine learning (ML) based techniques (*e.g.*, CrowdStrike, Endgame, Bkav, *etc*.). Machine learning techniques are widely viewed as one of the major viable solutions to combat the ever-growing big data problem, as more and more potentially malicious samples were collected every day. Previous work [2], [3], [4], [5], [6] in academia has repeatedly demonstrated that machine learning techniques can perform well [7] with prepared ground truth testing datasets.

To create an effective machine learning based malware detection (MLMD) system, researchers can follow a standard procedure. Firstly, they need to prepare a large set of ground truth data. For example, by scanning the collected samples against a list of existing Antivirus (AV) vendors. A sample is considered as *confirmed benign* if no vendor detected it as malicious and considered as *confirmed malicious* if the number of vendors detecting it as malicious was larger than a predefined threshold. The confirmed samples are then divided into disjoint training and testing datasets. Secondly, they need to extract various representative features from the ground truth training dataset and build an ML classifier. Finally, to examine the effectiveness of the newly designed system, the ML classifier will be evaluated against the ground truth testing dataset. We refer to the above general process as the `traditional approach` for creating and evaluating MLMD system.

Despite the seemingly encouraging results reported by previous research, the MLMD systems designed with traditional approach do not significantly improve malware detection experiences from end user perspective, *e.g.*, false positives are still common in malware detection domain and new AV solutions are continuously added [8] every year. If we define malware knowledge of a particular family as whether we can correctly detect the samples from that family, then the traditional approach does not dramatically contribute new malware knowledge to security community either, since the testing malicious samples were already confirmed by existing AV vendors. It is widely acknowledged that evaluation with confirmed malware samples (as shown in traditional approach) is absolutely necessary, or else we won't know how good or bad the newly designed malware detection system performs. However, such evaluation does not reflect the general strengths (*e.g.*, signals of maliciousness) and weaknesses (*e.g.*, lack of specificity) of MLMD systems, and ignores the reality that we already have a huge list of well-performing AV products.

Our evaluation is aligned with how an ML algorithm can potentially benefit malware detection in practice, by acknowledging the fact that any ML classifier has to be trained on potentially imperfect knowledge about apps' maliciousness, and the reality that we already have a huge list of working AV vendors which were used to create the ground truth labels. This evaluation seeks to answer 1) can an ML algorithm be helpful in producing new knowledge or value, even if trained on imperfect knowledge, and 2) what are the differences among a number of ML approaches with regard to their capability of generating the new knowledge. Specifically, we mainly examine the traditional prediction based malware detection. Results of experiments show that both approaches can be used to achieve verifiable zero-day malware detection, if we combine the ML outputs with existing AV detection capabilities. This indicates that MLMD system can produce

provable new malware knowledge to the security community when used together with existing AV vendors.

In summary, our major contributions are as follows:

- We design a new strategy to evaluate ML system outputs by checking the detection of previously unknown malware knowledge. This new evaluation strategy avoids the pitfalls identified within traditional approaches and illustrates an ideal usage scenario for MLMD systems.
- We demonstrate that MLMD system can consistently generate new malware knowledge even if trained with imperfect ground truth labels. It also indicates that a new MLMD system can be complementary to existing malware detection solutions and shows the real value of ML-based malware detection system in practice.

II. Motivation

Machine learning techniques have been frequently used for malware detection. In traditional approach, the training and testing datasets are completely separated from each other but prepared at the same time using the same level of malware knowledge, *e.g.*, they were simultaneously obtained based on scanning results from the same AV vendors. An ML model is then created on the training dataset and evaluated on the disjointed testing dataset, as shown in Figure 1.

Fig. 1: Simplified Overview of Traditional Approaches

Even though it is easy to create new MLMD systems following the traditional approach, we noticed that there are various pitfalls. It fails to reflect the true performances of such systems when applied in real-world, and does not demonstrate the real added values if researchers decided to create yet another new MLMD system. The overall procedures for creating an ML-based desktop malware detection system and an ML-based Android malware detection system are very similar, and the pitfalls we describe in this section are applicable to both domains. Thus, we do not address specific malware analysis issues, such as desktop malware obfuscation or Android repackaging, and the various pre-processing and feature extraction issues. In this work, we will mainly conduct experiments with Android application samples.

A. Pitfalls of Traditional Approach

The first pitfall is that MLMD technique is not suitable to be used as a standalone malware detection system that directly exposes detection results to end users. The majority of such systems were built as a two-class classifier, which can only predict the target sample as malicious or benign. In practice, only providing the maliciousness outputs to end users makes it hard to act upon receiving the "malicious" alerts. For example, we frequently see the following generic labels reported by various AV products: *malicious (high confidence)*, *UnclassifiedMalware*, *Suspicious_GEN*, *malicious confidence 90%*, *Artemis!XXX*, *PUA (Potentially Unwanted Application)*, *suspected of Trojan*, and *Trojan.AndroidOS.Generic*. Particularly, it is very ambiguous for end users to interpret such labels, *e.g.*, a "confidence 60%" label and a "confidence 90%" label does not make much difference, since both of them are not affirmative and it will be challenging for end users to decide whether to trust such detection or not. Additionally, such generic malicious labels shift the burden of malware verification to end users and dramatically reduce the usability of malware detection systems, especially in an enterprise environment. This does not mean MLMD systems are useless in practice, but that we shouldn't use and evaluate them in the same way as the classical (*e.g.*, signature-based) malware detection systems.

Another pitfall derives from the way how MLMD systems were evaluated. In particular, previous ML-based malware detection systems were mainly evaluated by testing against known malware samples that were confirmed by existing AV vendors at a specific time. However, we observe that evaluating with known malware creates a dilemma for demonstrating the practical utility of the MLMD systems. On one hand, the correctness of the malware detection results needs to be verified by checking against the ground truth references. On the other hand, if the testing apps are known malware (*i.e.*, already detectable by existing AV products), such evaluation only shows that the proposed malware detection system can reproduce the same detection knowledge of existing AV products, instead of demonstrating detection knowledge superior to them.

In addition, the evaluation results of an MLMD system are often compared against individual existing AV vendors. Such comparison results are very deceptive, since the training dataset was created by using existing AV vendors' detection capabilities. For example, researchers often use the majority voting information provided by existing AV vendors to prepare a large set of ground truth training data. Training with such dataset means the proposed ML system already utilizes the intrinsic malware detection knowledge derived from existing AV products, thus it will be no surprise to see that the trained ML model (with combined malware knowledge) has better detection results than any individual product. Such evaluation also gives a false indication that the MLMD system was designed to compete and replace existing ones, which is not the case in reality (*e.g.*, more and more malware detection systems are added to VirusTotal) and does not demonstrate the strength of ML-based malware detection system.

The last pitfall is that the ground truth datasets used for training and testing usually are usually the easy to distinguish cases and will never be representative towards the real-world situation. The benign and malicious labels for the training dataset are created using existing AV scanning results, which may be highly unreliable [9], [10], and heavily depend on AV products' detection capabilities at a specific time. In practice, it is difficult to completely eliminate label noises from a training dataset, especially for the benign labels in the training

dataset if zero-day malware exists. Such "mislabels" in the data represent the imperfect knowledge security community has at the training time, and it will impact the quality of the classifier created from it. The target samples in real-world will definitely be more challenging to analyze than those included in the well-labeled testing dataset, *e.g.*, more borderline cases. Therefore, evaluation with well-labeled testing datasets (as in traditional approach) produces misleading results.

B. Real Values of MLMD System

As discussed in the previous section, there are various pitfalls for following the traditional approach to create MLMD systems. Fundamentally, there is a mismatch between how the ML system has been evaluated in the traditional approach, and the true purpose of using ML system in practice. The goal of applying ML is not to reproduce or verify the same potentially imperfect malware knowledge, but rather to produce something that is better — closer to the ultimate ground truth about the apps' maliciousness.

If malware is already flagged by AV products, there is not much use for an ML classifier to tell us the same thing. Thus, the ideal and meaningful application of an ML-based malware classifier in practice is to triage apps that are yet to be flagged by any existing AV products. For example, a security company may receive a large number of suspicious apps on a daily basis, which are not known by any of the AV products to be malicious. An ML-based approach should be used to predict the likelihood the apps may be new malware (excluding those that are also detected by existing AV vendors), so precious human analyst time can be prioritized to the more likely novel malware samples.

In this application setting, the value of an ML-based classifier lies in whether it can generate new malware knowledge, *e.g.*, malware that has yet to be flagged by any AV products. That is, to evaluate the effectiveness of an ML approach for malware triaging, we shall test the classifier on samples that are not yet known to be malware at the time the classifier is trained. Specifically, for the problem of Android malware detection, if a classifier is trained based on some existing knowledge, its utility can only be demonstrated by how well it can predict that an app, not yet known by the existing knowledge to be malware, is actually malware.

III. Alternative Evaluation Approaches

Observing the mismatch between how ML systems have been evaluated in previous malware detection research and the true purpose of malware detection in practice, we argue that the usefulness of a real-world machine learning system is not in verifying or reproducing what has already been known (*e.g.*, confirmed malware). Rather, it is most helpful if it can be used to produce verifiable results that are previously unknown (*e.g.*, zero-day malware), which arguably is the very goal of using machine learning techniques in this problem domain.

Therefore, we proposed to evaluate machine learning outputs by checking **zero-day malware detection** with a collection of old Android samples. Real-world malware detection is very time-sensitive and heavily relies on existing knowledge about the malware family. A malware app is a zero-day malware if no AV products can detect it at a specific time t. Once the malware has been studied, it becomes known to the public as malicious and the malware shouldn't be considered as zero-day anymore. Note that the zero-day malware described in this paper is not exactly the same as the zero-day concept well-known in the security community, which often refers to those samples that no AV products can detect at the current time.

We prepared two sets of ground truth knowledge labels to confirm the correctness of real zero-days. For example, given a large number of Android apps, we scan them against VirusTotal at time t_1 to get the first set of ground truth labels, and scan them at time $t_1 + \delta$ to get the second set of ground truth labels for the same dataset. The intuition is that if we wait a period of time and rescan the samples again, we may find that some apps' scanning results changed from benign to malicious due to AV vendors' updated malware detection capabilities, indicating that the app was a zero-day malware earlier but later detected as malicious by the AV vendors. Therefore, the second set of labels have higher quality and would be closer to the ultimate ground truth about apps' maliciousness.

For the set of malicious samples that were detected by MLMD system, we compare them against the first set of ground truth labels to identify the list of `potential zero-days`; then compare the potential zero-days against the second set of ground truth labels to obtain the list of `confirmed zero-days`. By examining old Android samples and their corresponding evolving AV scanning labels, we can retrospectively inspect whether an ML-based system can identify zero-day malware at an earlier time, *i.e.*, if we conduct the in-depth analysis with the potential zero-days at time t_1, we could have identified those real zero-days by then.

A. Malware Prediction

Existing ML-based malware detection systems [2], [3], [4], [5], [6], [7] are commonly referred to as malware classification or **malware prediction**. During the malware prediction process, an ML model is built upon the provided ground truth training dataset, and the ML model is then used to analyze the target samples and generate a prediction for each sample indicating its maliciousness.

In addition to the well-known ML algorithms, we describe a special algorithm called `Label Regularized Logistic Regression` (LR-LR) which can be used for malware prediction scenario. The LR-LR approach is a semi-supervised variant of Logistic Regression, training over only positive and unlabeled data. It was recently applied to Android malware detection as a means for potentially detecting inaccuracies in existing VirusTotal-labeled ground truth [11]. Instead of using the benign class as negative examples and attempting to learn the difference between positive and negative classes, LR-LR trains upon the positive data, and then regularizes the class distribution in the unlabeled data towards an expected distribution. Specifically, an expert-provided $\tilde{p}$ constant is used, which represents the expected proportion of positives

in the unlabeled dataset. The optimization function then regularizes the current expectation of positives in the unlabeled class into what it should be, as defined by $\tilde{p}$. Eventually, a classifier that can predict positives and negatives is trained only upon positive data, unlabeled data, and an expectation of how many positives exist in the unlabeled set. The novelty of the approach is that it allows utilization of benign data as unlabeled data, thus avoiding relying upon benign labels which are inevitably noisy.

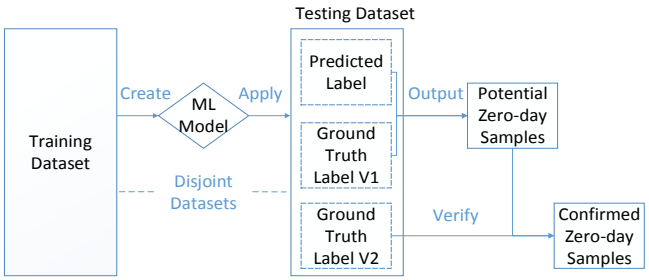

Fig. 2: Malware Prediction Experiment Overview

We show the experiment overview of using malware prediction techniques for zero-day malware detection in Figure 2. The major characteristic for the malware prediction approach is that the ML model is trained and applied on disjoint datasets. In order to show verifiable zero-day detection results, we use the first set of ground truth labels to get the potential zero-day detection results and use the second set of ground truth labels to verify the zero-day detection results.

B. Noise Identification

In practice, we can create a cleaner malicious training dataset by tuning the maliciousness decision threshold. For example, only consider the samples as malicious if they are detected by at least n different AV products, and exclude the rest cases. But for the benign dataset, there is only one way to prepare the training dataset: considering all the samples that are not detected as malicious by any AV products to be benign. This may result in poor performing ML models if the benign training dataset contains noise (*e.g.*, the zero-day samples). Therefore, we also consider another category of machine learning techniques called **mislabel identification**, which can be used to identify and remove mislabels in the training dataset.

We describe a special mislabel identification approach named `Active Label Noise Removal` (ALNR) which has been proposed recently in the machine learning community. ALNR is a two-stage process to identify the label noise from the training dataset. This approach is built on top of the SVM [12] algorithm and designed [13] based on the observation that the maximum margin principle used in SVM has the characteristic of capturing the mislabeled samples as support vectors; thus the label noise examples contained in the training dataset will likely be selected as support vectors for the trained SVM classifier. In `Stage-1`, ALNR uses all the provided malware and benign samples to build a standard SVM classifier `A` and finds the support vectors for the provided dataset. In `Stage-2`, it removes the support vectors (SVs) and uses all the remaining non-SV samples to build a new classifier B. Then, it uses classifier B to classify the benign SVs identified from Stage-1. If B's predicted labels are not the same as the original provided benign labels, the algorithm reports the sample as a mislabel in the benign dataset.

IV. Experiment Preparation

We collected Android apps from various sources, such as PlayDrone [14], AndroZoo [15], VirusShare [16], internet service providers, and security companies. The apps were separated into the following three major categories: scanned malicious samples, scanned benign samples, and unknown samples.

A. Malware Samples

We used AV scanning results to prepare the malware dataset. To obtain AV scanning results, we query the MD5 of the samples against VirusTotal, which is an online service that integrates more than 50 different AV scanners. Particularly, we consider a sample detected by at least n different AV products as *confirmed* malicious, a sample detected by less than n number of AV products as *unconfirmed*.

In this work, we decided to use the AV detection threshold of 10 to prepare the malicious training dataset, since the scanning results for such malware samples are typically very stable. We used the AV scanning reports obtained in April 2016 to prepare the malware dataset, and randomly selected 15000 malicious samples (*i.e.*, referred to as `M-2016`) from all available malicious samples. We didn't use all available malicious samples because: (1) it will take a significantly longer time to conduct the experiment using all the samples. (2) previous research [17] showed that the malware to benign class ratio of the training dataset could impact the performance of the trained classifier. (3) we decided to configure all of our experiments with a tractable class ratio (malware vs benign) around 1:10, to consider the unbalanced ratio of malware and benign apps in reality and avoid the potential bias towards an overwhelming number of benign samples.

We only use the confirmed malware samples for ML training purpose, since the main focus of the experiment is zero-day malware detection and the malicious training dataset is prepared with high-quality (*i.e.*, high AV detection threshold).

B. Benign Samples

A scanned benign sample means that the MD5 of the app was queried against VirusTotal, and no malicious flags were contained in its scanning report at the querying time. However, since scanning data for a particular app may be outdated, the existing scanning results may not represent the AV products' latest malware detection capabilities. Therefore, we conducted multiple rounds of scanning within the last four years to obtain the upgraded ground truth labels for the same dataset.

We used the scanning reports obtained before April 2016 as the resources for preparing our benign sample dataset and removed those samples that hadn't been analyzed by VirusTotal by April 2016. That is, all of the scan dates contained in those reports are earlier than April 2016, and none of the AV products had detected them as malicious as of April 2016. Finally, 811649 Android samples (*i.e.*, referred to as `B-2016`) were labeled as benign since no malicious label was returned for those samples. Note that we will use the malicious and benign samples labeled by April 2016 scanning results for training purpose.

For the same dataset, we again collected their scanning reports around December 2017. We observed that the majority of the scan dates for those samples were updated in 2017, which means they were rescanned in 2017. According to the newly obtained scanning reports, we observed 751804 of them were still detected as benign samples. Even though a substantial number of samples were changed from benign to malicious, we noticed that about 69% of them are only detected by one AV vendor, which means the newly assigned malicious labels may contain significant false positives.

C. Unknown Samples

We also selected unknown Android samples which were not scanned and queried against VirusTotal before MLMD training time (*i.e.*, April 2016). Particularly, we checked the timestamp of the main Dex file contained in each unscanned Android app, and only selected those whose timestamps were later than June 30, 2016. Thus, all of the unknown samples were newer Android apps compared with the labeled malware samples and benign samples. In the end, we obtained 413353 unknown Android samples (*i.e.*, referred to as Unknown `U`).

TABLE I: Summary of the Prepared Datasets

Dataset	Size	Label V*1*	Label V*2*
Malware **M-2016**	15000	Apr 2016	-
Benign **B-2016**	811649	Apr 2016	Dec 2017
Unknown **U**	413353	May 2017	Mar 2019

To sum up, we present the overall dataset in Table I. For the rest of the paper, we refer to the 15000 malware samples obtained using April 2016 AV scanning results as malware dataset **M-2016**, the 811649 benign samples obtained using April 2016 AV scanning results as benign dataset **B-2016**, and the unknown samples (as of June 30, 2016) as unknown dataset **U**. Note that the word "unknown" is a relative concept comparing with training datasets. To conduct the zero-day detection experiment with dataset `U`, we eventually scanned a subset of those that were detected as malicious by MLMD systems in May 2017 and March 2019.

D. Feature Construction

Since the focus of this work is not about feature construction, we decided to use previously known successful Android malware features. Particularly, we used the same set of Android app features as defined in [17], because the overall feature set described by them is relatively small, *e.g.*, compared with Drebin [7] which may end up with millions of unique features. Further, it contains the majority of Android app features that have been shown to work well in previous Android malware research [18], [19], [20].

To obtain the features, we conducted the lightweight static analysis for the Android apps, and extracted 471 different features which can be separated into the following categories: critical API usage, permission requests, intent action, obfuscation characteristics, native code signatures, *etc*. Note that the values for all of the features are limited to 1 and 0, which is used to indicate whether the app contains a particular feature or not. Since the features are already condensed, we do not further apply any feature selection process, *i.e.*, all subsequent experiments are conducted with the same set of feature vectors.

V. Malware Prediction (MP) Experiment

We design an experiment to check whether we can achieve verifiable zero-day malware detection using MP techniques. Verifiable means the samples were initially identified as benign and later confirmed as malicious, thus showing that the samples were real zero-day malware at the time when they were detected by the ML models.

In this experiment, we used one popular classical machine learning algorithm—Random Forests (RF)—as the first representative MP algorithm, since it is efficient and can usually generate good results without much tuning effort. We used the Label Regularized Logistic Regression (LR-LR) as the second MP algorithm since its ML model does not rely on a benign training dataset. Because mislabel identification techniques can be used to remove mislabels, we also used the Active Label Noise Removal (ALNR) algorithm to firstly remove the potential mislabels from the benign training dataset, then conducted the MP experiment with the RF approach. For simplicity, we call the RF based malware prediction approach as RF_MP, call the LR-LR based malware prediction approach as LR-LR_MP, and call the (benign) mislabels-removed malware prediction with RF approach as ALNR+RF_MP.

A. Testing with Benign `B-2016` Dataset

In this section, we show the MP experiment using the benign `B-2016` dataset as the testing dataset. Since the MP approach needs to train and test on the disjoint datasets, we used `M-2016` and part of `B-2016` for training and used the rest of `B-2016` for testing. In order to reduce the training size and create a training dataset with an approximate malware to benign class ratio of 1:10, we divided the benign dataset `B-2016` into 5 folds, then trained a classifier using malware dataset `M-2016` and each fold of `B-2016` and tested the classifier with the remaining folds of `B-2016`, hence each sample in `B-2016` was predicted 4 times. Eventually, we ensembled all of the predictions using a threshold of 2. Figure 3 shows the overall training and testing data composition of this experiment. For each MP approach, there will be 5 different ML models built to get the all-fold predictions.

We conducted parameter selection for each MP approach using another set of training data with a similar number of

TABLE II: Confirmed zero-day malware detection results for B-2016 dataset using malware prediction techniques

Approach	Predicted Malware	Potential Zero-days	Detection threshold *n=10*		Detection threshold *n=5*		Detection threshold *n=2*	
			Confirmed	Unconfirmed	Confirmed	Unconfirmed	Confirmed	Unconfirmed
RF_MP	1561	1561	457	1104	627	934	791	770
LR-LR_MP	12093	12093	490	11603	815	11278	1715	10378
ALNR+RF_MP	2174	2174	512	1662	753	1421	1073	1101

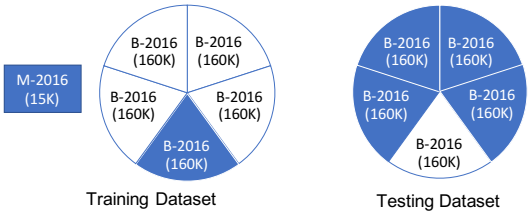

Fig. 3: Setup for one round of MP experiment with B-2016

samples as the formal experiment. For example, we randomly sampled 1/5 of the data from B-2016 as benign training data and used this sampled benign data and all malware M-2016 data for parameter selection. We used the grid search approach with 5-fold cross-validation to conduct parameter selection for all ML approaches. The number of trees for RF was 200 with *sqrt* number of features; the $\tilde{p}$ (expected proportion of positives in the unlabeled dataset) and λ_U (label regularization parameter) parameters for LR-LR were 0.1 and 3, respectively; the C parameter for standard SVM with linear kernel [1] was 0.5, which was used by the two classifiers in ALNR algorithm.

Since all samples in B-2016 were benign according to the April 2016 VirusTotal scanning results, all of the predicted malicious samples on B-2016 can be viewed as potential zero-days (as of April 2016). We considered the December 2017 VirusTotal scanning reports for the B-2016 dataset as the upgraded ground truth labels and used the new ground truth labels to evaluate the zero-day detection results. The overall zero-day malware detection results for different malware prediction approaches are shown in Table II, where each entry represents the ensemble (*e.g.*, all-fold) results for the corresponding approach. From the table, we can see that all MP approaches detected a substantial amount of potential and confirmed zero-day malware samples. And cleaning the training dataset by removing the potential mislabels (*e.g.*, using ALNR) can help detect more confirmed zero-day malware.

Comparing with previously reported Android malware detection results with regard to the detection of confirmed malware, the accuracy for zero-day malware detection is relatively low. However, one must consider that we are training with malware knowledge only obtainable from April 2016, when none of the AV products was able to detect any of the later confirmed zero-day malware at that time. We thus view it as a positive result, since it not only shows an ML-based system can achieve verifiable zero-day malware detection, but also reveals the challenging reality for the problem, *e.g.*, an analyst may need to conduct in-depth analysis for a substantial number of candidates to identify the real zero-days.

In summary, this experiment shows that the MP-based detection approaches indeed detected a substantial number of zero-day samples which were confirmed by other AV products about one and a half years later. For example, if considering the samples that were detected by at least 2 AV products as "confirmed" malware, then about 50% of the potential zero-days were indeed malicious for RF_MP approach and ALNR+RF_MP approach. The LR-LR approach generated a higher number of confirmed zero-day samples, but also produced a dramatically larger number of unconfirmed detections, which results in lower detection accuracy.

B. *Testing with Unknown* U *Dataset*

We perform the MP based zero-day detection experiment with the unknown U dataset in this section. The high-quality malware dataset M-2016 and benign dataset B-2016 were used for training, and the samples in unknown dataset U were used for testing. Figure 4 illustrates the training and testing setup for this experiment. We used the same data split for training datasets and the same set of optimal parameters as obtained in Section V-A. We first trained a classifier using the malware dataset M-2016 and each fold of B-2016, then tested the classifier with all the samples in U. In summary, each sample in the unknown dataset U was predicted 5 times since there were 5 different ML models for each MP approach. Finally, we ensembled the predictions using a threshold of 2 to get the aggregated detection results.

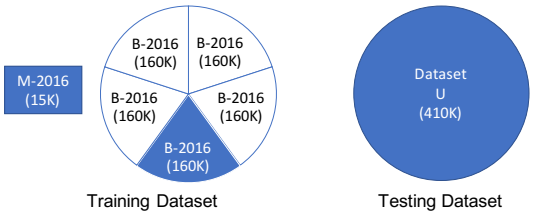

Fig. 4: Setup for one round of MP experiment with U

For the predicted malicious samples in Dataset U, we first checked their May 2017 AV scanning results to get the potential zero-days, then examined their March 2019 AV scanning results to obtain the confirmed zero-day malicious samples. The overall zero-day detection results for unknown dataset U using MP approaches are shown in Table III. Even though all MP approaches detected significant number of malware in U, a large number of the predicted malware were already confirmed as malicious by existing AV products by May 2017.

[1]Other kernels were not used since they were not as efficient as linear kernel and their performance improvements were very small

TABLE III: Confirmed zero-day malware detection results for U dataset using malware prediction techniques

Approach	Predicted Malware	Potential Zero-days	Detection threshold *n=10*		Detection threshold *n=5*		Detection threshold *n=2*	
			Confirmed	Unconfirmed	Confirmed	Unconfirmed	Confirmed	Unconfirmed
RF_MP	16179	2221	30	2191	69	2152	200	2021
LR-LR_MP	31366	23482	47	23435	157	23325	463	23019
ALNR+RF_MP	20963	3206	39	3167	92	3114	286	2920

If we remove those malware that were already detectable by May 2017, all MP approaches still have about 10% of the potential zero-days got confirmed in March 2019. This experiment shows MP approaches can generate new malware knowledge (*e.g.*, zero-days as of May 2017) even if trained with old (April 2016) labeled samples. Apparently, prioritizing the in-depth analysis within the potential zero-days will be more rewarding than analyzing all samples in U.

VI. Discussion

A. The Ideal MLMD Usage Scenario

To confirm zero-day detection during the experiments, we obtained two sets of ground truth labels for the same dataset. In reality, there is no value to scan the dataset with AV products, wait for a certain period of time then scan it again to obtain the set of confirmed "zero-day" malware. The second stage analysis is to demonstrate that we can employ in-depth analysis on a small set of samples that are detected by MLMD system to identify verifiable zero-days (*i.e.*, new malware knowledge). In practice, such in-depth analysis could be manual analysis or dynamic analysis techniques that can extract more detailed behaviors or activities but are often more time-consuming to conduct and maybe impossible to apply for large scale dataset.

Therefore, we want to emphasize that this work is not to design an MLMD system for zero-day detection. Rather, we want to explore the real values of MLMD systems such that the newly designed system can be meaningfully evaluated and effectively applied in practice. As illustrated by the experiments with the unknown testing dataset, we can filter out significant number of malicious samples that are already detectable by existing AV vendors, and only conduct the in-depth analysis within the remaining potential zero-days. In this way, we utilize the strengths of both the existing AV vendors and the newly designed MLMD system. We believe this indicates a more ideal utilization of the MLMD systems in practice.

This usage demonstrates an effective malware triaging process. It is particularly helpful for handling large scale potentially malicious samples within security companies, AV vendors, or application stores. On one hand, this process is perfectly aligned with practical malware triaging scenario, avoids the major pitfalls discussed in Section II-A; On the other hand, it allows us to convert the tacit malware knowledge (*e.g.*, various feature combinations) captured by ML models to more concrete malware knowledge (*e.g.*, specific malware signatures) in traditional AV vendors, which in turn leads to more accurate ground truth dataset, helps to improve the ML models.

B. False Positives and False Negatives

As shown in Section V-B, even if we use the latest AV scanning results to evaluate the MLMD system outputs, there are still lots of unconfirmed zero-days, those samples can be viewed as false positives of the zero-days. And depending on ML algorithm and parameter configuration, a large portion of MLMD outputs may be false positives, from which we won't be able to derive new malware knowledge. Thus, we will need to analyze a large number of potential zero-days to confirm the new malware knowledge. Due to the lack of specificity for ML models, false positives are part of the expected and inevitable outputs for all ML-based detection systems. And it would be better if we can handle them from the vendor side before directly exposing such results to end users.

At the same time, different MLMD systems generated different sets of confirmed zero-days. Admittedly, each individual ML-based malware detection system has a certain number of false negatives of the real zero-days. Since the malware knowledge is consistently evolving for all AV vendors, we may never obtain the ultimate ground truth for the evaluated datasets, *e.g.*, it's possible that some real zero-days still contained in dataset U that none of the existing AV vendors can detect due to their highly obfuscated code or low-profile malicious activities. Therefore, it is almost impossible to precisely quantify the zero-day detection accuracy and the real ratio of false positives and false negatives. Because of this, the various standard measurements (*e.g.*, FPR, TPR, precision, recall, *etc.*) adopted by traditional approaches are less meaningful in the context of zero-day detection. Thus, we can only obtain MLMD systems' relative performances by checking the number and ratio of confirmed zero-day detection for the same dataset.

Nevertheless, by confirming that MLMD systems can help to identify new malware samples that were not detected by any of the existing AV vendors at the training time, we showed the real value of MLMD systems. In addition, we can employ various strategies to reduce false positives and false negatives. Firstly, we can create new MLMD systems by choosing the combination of different features, ML algorithms, and malware prediction or mislabel identification approaches. In such manner, a new MLMD system may help to identify the real zero-days that are missed by other systems. Secondly, we can apply ensemble techniques to combine multiple MLMD systems together and prioritize the potential zero-days that are detected by more than one MLMD system or have higher prediction confidences.

VII. Related Work

Various machine learning based systems [7], [21], [19], [20] have been proposed for Android malware detection. Drebin [7]

gathered a massive set (more than 500K) of features which contain different types of manifest features (*e.g.*, permissions) and "code" features (*e.g.*, URLs, APIs). It uses an SVM to train a detection model which later can be uploaded to a device to do on-device malware detection. Drebin's performance results are impressive. MUDFLOW [21] discovered that the sensitive information flow pattern is different between benign and malware apps which can be utilized to do malware detection. The data flow information is then used as features in a standard SVM to train classifiers. MAST [19] helps resource intensive operations (*e.g.*, manual analysis) to triage their priority, thereby reducing the average computation overhead. This system utilizes a statistical method called Multiple Correspondence Analysis (MCA), and uses permissions, intents and the presence of native code to determine the probabilities of being malicious. DroidSIFT [20] builds the API dependency graphs for each app and uses the graph as the feature vector. Then the feature vectors are used to train a classifier to do anomaly or signature detection.

Unlike these prior works which train and test their approaches both on a "known" ground truth dataset, we start with the different goal of finding the "unknown" knowledge (*i.e.*, zero-day malware), and systematically show that the various machine learning approaches can achieve real-world zero-day malware detection.

VIII. Conclusions

In this paper, we conducted an observational study using a collection of old Android samples and several sets of evolving maliciousness labels obtained during the past several years. To demonstrate the real value of new MLMD systems, we proposed a new strategy to evaluate ML system outputs by checking the detection of previously unknown malware samples. Through comprehensive evaluation, we confirmed that MLMD systems can consistently generate new malware knowledge, even if trained with imperfect ground truth labels. The evaluation illustrates an ideal MLMD usage scenario for malware triaging in practice. It also demonstrates that MLMD systems can be complementary to existing malware detection solutions. For instance, we could reliably uncover new malware variants or families if we conduct in-depth analysis within those samples that are detected by MLMD systems but reported as benign by existing AV vendors.

Acknowledgment

This research is partially supported by the National Science Foundation under Grant No. 1622402, 1717862, and 1717871. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] "Virustotal - free online virus, malware and url scanner," https://www.virustotal.com, 2017.

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

[3] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in *Computational Intelligence and Security (CIS), 2010 International Conference on*. IEEE, 2010, pp. 329–333.

[4] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639–668, 2011.

[5] J. Sahs and L. Khan, "A machine learning approach to android malware detection," in *Intelligence and security informatics conference (eisic), 2012 european*. IEEE, 2012, pp. 141–147.

[6] N. Peiravian and X. Zhu, "Machine learning for android malware detection using permission and api calls," in *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*. IEEE, 2013, pp. 300–305.

[7] 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*, 2014.

[8] "Two-thirds of all android antivirus apps are frauds," https://zd.net/2JfFaMQ, 2019.

[9] M. Sebastián, R. Rivera, P. Kotzias, and J. Caballero, "AVclass: A tool for massive malware labeling," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2016, pp. 230–253.

[10] Y. Le Traon, "On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware," *Detection of Intrusions and Malware, and Vulnerability Assessment: DIMVA*, p. 142, 2016.

[11] J. DeLoach, D. Caragea, and X. Ou, "Android malware detection with weak ground truth data," in *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 2016, pp. 3457–3464.

[12] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[13] R. Ekambaram, S. Fefilatyev, M. Shreve, K. Kramer, L. O. Hall, D. B. Goldgof, and R. Kasturi, "Active cleaning of label noise," *Pattern Recognition*, vol. 51, pp. 463–480, 2016.

[14] N. Viennot, E. Garcia, and J. Nieh, "A measurement study of google play," in *ACM SIGMETRICS Performance Evaluation Review*, vol. 42. ACM, 2014, pp. 221–233.

[15] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *Proceedings of the 13th International Conference on Mining Software Repositories*. ACM, 2016, pp. 468–471.

[16] "Virusshare.com," https://virusshare.com/, 2017.

[17] S. Roy, J. DeLoach, Y. Li, N. Herndon, D. Caragea, X. Ou, V. P. Ranganath, H. Li, and N. Guevara, "Experimental study with real-world data for android app security analysis using machine learning," in *Proceedings of the 31st Annual Computer Security Applications Conference*. ACM, 2015, pp. 81–90.

[18] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*. IEEE, 2012, pp. 62–69.

[19] S. Chakradeo, B. Reaves, P. Traynor, and W. Enck, "MAST: Triage for market-scale mobile malware analysis," in *Proceedings of the sixth ACM conference on Security and privacy in wireless and mobile networks*. ACM, 2013, pp. 13–24.

[20] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 1105–1116.

[21] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden, "Mining apps for abnormal usage of sensitive data," in *Proceedings of the 37th International Conference on Software Engineering-Volume 1*. IEEE Press, 2015, pp. 426–436.