securityandtrust.lu

Machine Learning-Based Malware Detection for Android Applications: History Matters!

Kevin Allix	University of Luxembourg / SnT / SERVAL, Luxembourg
Tegawendé F. Bissyandé	University of Luxembourg / SnT / SERVAL, Luxembourg
Jacques Klein	University of Luxembourg / SnT / SERVAL, Luxembourg
Yves Le Traon	University of Luxembourg / SnT / SERVAL, Luxembourg

26 May 2014

978-2-87971-132-4

Machine Learning-Based Malware Detection for Android Applications: History Matters!

Kevin Allix
SnT, University of Luxembourg

Tegawendé F. Bissyandé
SnT, University of Luxembourg

Jacques Klein
SnT, University of Luxembourg

Yves Le Traon
SnT, University of Luxembourg

Abstract

Machine Learning-based malware detection is a promising scalable method for identifying suspicious applications. In particular, in today's mobile computing realm where thousands of applications are daily poured into markets, such a technique could be valuable to guarantee a strong filtering of malicious apps. The success of machine-learning approaches however is highly dependent on (1) the quality of the datasets that are used for training and of (2) the appropriateness of the tested datasets with regards to the built classifiers. Unfortunately, there is scarce mention of these aspects in the evaluation of existing state-of-the-art approaches in the literature.

In this paper, we consider the *relevance of history* in the construction of datasets, to highlight its impact on the performance of the malware detection scheme. Typically, we show that simply picking a random set of known malware to train a malware detector, as it is done in most assessment scenarios from the literature, yields *significantly biased* results. In the process of assessing the extent of this impact through various experiments, we were also able to confirm a number of intuitive assumptions about Android malware. For instance, we discuss the existence of Android malware lineages and how they could impact the performance of malware detection in the wild.

1 Introduction

Malware detection is a challenging endeavor in mobile computing, where thousands of applications are uploaded everyday on application markets [5] and often made available for free to end-users. Market maintainers then require efficient techniques and tools to continuously analyze, detect and triage malicious applications in order to keep the market as clean as possible and maintain user confidence. For example, Google has put in place a number of tools and processes in the Google Play official market for Android applications. However, using antivirus software on large datasets from Google reveals that hundreds of malware are still distributed incognito through this market.

Unfortunately, malware pose various threats that cannot be ignored by users, developers and retailers. These threats range from simple user tracking and leakage of personal information [21], to unwarranted premium-rate subscription of SMS services, advanced fraud, and even damaging participation to botnets [33]. To address such threats, researchers and practitioners increasingly turn to new techniques that have been assessed in the literature for malware detection in the wild. Research work have indeed yielded promising approaches for malware detection. A comprehensive survey of various techniques can be found in [26]. Approaches for large-scale detection are often based on Machine learning techniques, which allow to sift through large sets of applications to detect anomalies based on measures of similarity of features [7, 12, 17, 24, 29, 32, 36, 39, 44].

To assess malware detection in the wild, the literature resorts to the 10-Fold Cross validation scheme with datasets that we claim are biased and yield biased results. Indeed, various aspects of construction of training datasets are usually overlooked. Among such aspects is the *history aspect* which assumes that the training dataset, which is used for building classifiers, and the test dataset, which is used to assess the performance of the technique, should be *historically coherent*: the former must be historically anterior to the latter. This aspect is indeed a highly relevant constraint for real-world use cases and we feel that evaluation and practical use of state-of-the-art malware detection approaches must follow a process that mimics the history of creation/arrival of applications in markets as well as the history of appearance of malware: *detecting malware before they are publicly distributed in markets is probably more useful than identifying them several months after they have been* *made available*.

Nevertheless, in the state-of-the-art literature, the datasets of evaluation are borrowed from well-known labelled repositories of apps, such as the Genome project, or constructed randomly, using market-downloaded apps, with the help of Antivirus products. However, the history of creation of the various apps that form the datasets are rarely, if ever, considered, leading to situations where **some items in the training datasets are "*from the future*", i.e., historically posterior to items in the tested dataset**. Thus, different research questions are systematically eluded in the discussion of malware detector performance:

RQ-1. Is a randomly sampled training dataset equivalent to a dataset that is historically coherent to the test dataset?

RQ-2. What is the impact of using malware knowledge "from the future" to detect malware in the present?

RQ-3. How can the potential existence of families of malware impact the features that are considered by machine learning classifiers?

RQ-4. How *fresh* must be the apps from the training dataset to yield the best classification results?

RQ-5. Is it sound/wise to account for all known malware to build a training dataset?

RQ-6. Finally, what are the steps to take towards building a reliable training dataset?

This paper. We propose in this paper to investigate the effect of ignoring/considering historical coherence in the selection of training and test datasets for malware detection processes that are built on top of Machine learning techniques. Indeed we note from literature reviews that most authors do not take this into account. Our ultimate aim is thus to provide insights for building approaches that are consistent with the practice of application –including malware– development and registration into markets. To this end, we have devised several typical machine learning classifiers and built a set of features which are textual representations of basic blocks extracted from the Control-Flow Graph of applications' byte-code. Our experiments are also based on a sizeable dataset of about 200,000 Android applications collected from sources that are used by authors of contributions on machine learning-based malware detection.

The contributions of this paper are:

- We propose a thorough study of the history aspect in the selection of training datasets. Our discussions highlight different biases that may be introduced if this aspect is ignored or misused.
- Through extensive experiments with tens of thousands of Android apps, we show the variations that the choice of datasets age can have on the malware detection output. To the best of our knowledge, we are the first to raise this issue and to evaluate its importance in practice.
- We confirm, or show how our experiments support, various intuitions on Android malware, including the existence of so-called lineages.
- Finally, based on our findings, we discuss (1) the assessment protocols of machine learning-based malware detection techniques, and (2) the design of datasets for training real-world malware detectors.

The remainder of this paper is organized as follows. Section 2 provides some background on machine learning-based malware detection and highlights the associated assumptions on dataset constructions. We also briefly describe our own example of machine-learning based malware detection. Section 3 presents related work to support the ground for our work. Section 4 describes the experiments that we have carried out to answer the research questions, and presents the take-home messages derived from our empirical study. We propose a final discussion on our findings in Section 5 and conclude in Section 6.

2 Preliminaries

The Android mobile platform has now become the most popular platform with estimated hundreds of thousands of apps in the official Google Play market alone and downloads in excess of billions. Unfortunately, as this popularity has been growing, so is malicious software, i.e., malware, targeting this platform. Studies have shown that, on average, Android malware remain unnoticed up to 3 months before a security researcher stumbles on it [6], leaving users vulnerable in the mean time. Security researchers are constantly working to propose new malware detection techniques, including machine learning-based approaches, to reduce this 3- months gap. The following sub-sections present the main concepts upon which a machine learning approach is based. Moreover, we describe our dataset and our overall methodology.

2.1 Malware Detection in the wild

In February 2012, Google has introduced the Bouncer [23] which leverages on dynamic analysis to identify malware based on execution behavior. The current rates of malware appearing in Google Play demonstrates however that this analysis can be circumvented. Besides, a number of third party Android markets are currently growing, most of which might take less steps to scrutinize apps before upload.

To better devise techniques and tools for fighting malware, researchers must have at disposal an extensive knowledge on existing malware. Unfortunately, spotting malicious samples in Android is also difficult [6]: security practitionners have hard time collecting all Android apps and end-users are not always aware of malicious activities that need reporting. To detect zero-day malware attacks, i.e., attacks that were previously unknown to the malware detector, researchers resort to anomaly-based detection schemes [26]. They occur in two phases: a training phase during which the detector attempts to learn to discriminate between the normal and abnormal behavior/attributes, and a detection phase. Also referred to as machine learning-based detection schemes, these approaches are known to present two limitations: they have high false alarm rates and determining what features should be learned in the training phase is a complex endeavour. A key step in these approaches thus resides in the process of selecting datasets for training.

2.2 Machine Learning

As summarized by Alpaydin, "Machine Learning is programming computers to optimize a performance criterion using example data or past experience" [3]. A common method of learning is known as *supervised* learning, a scheme where the computer is helped through a first step of *training* (as with a teacher). The training data consists of Feature Vectors, each associated with a label, e.g., in our case, apps that are already known to be malicious (*malware* class) or benign (*goodware* class). After a run of the learning algorithm, the output is compared to the target output and learning parameters may be corrected according to the magnitude of the error. Consequently, to perform a learning that will allow a *classification* of apps into the malware and goodware classes, the approach must define a correlation measure and a discriminative function.

Malware features & Training data To devise an effective similarity measure, one must first define what app metadata, or code, characteristics can provide the best indicators of potential malicious activities. Such *features* must be extracted and potentially weighted following their importance. Then, one must determine the threshold (or a much more complex measure) of features value, that must be reached to be classified as malware.

As one could expect, given the importance of the threshold and the kinds of features that will be retained as good indicators, the extent and quality of training dataset will drive all the assessment process. The training dataset must thus be carefully sampled. The literature of Android malware detection includes diverse examples of features, such as n-grams of source code, API usages, application permission uses, etc.

Machine Learning Algorithms & Other parameters The execution of the machine learning process starts with the construction of a classification model. The literature of malware detection often implement approaches leveraging one of the algorithms that are well-known in the community of machine learning. For instance, the Weka[1] framework provides, amongst many others, the RandomForest, J48, JRip and LibSVM implementations of, respectively, Support Vector Machine (SVM) [19], the RandomForest ensemble decision-trees algorithm [13], the RIPPER rule-learning algorithm [18] and the tree-based *C4.5* algorithm [35]. A second variable of the process is the extent of the class imbalance between goodware and malware in the training dataset. Finally, the volume of features considered to discriminate malware from goodware, also has an impact on the performance of the detection technique.

In our work, because we focus exclusively on the history aspect, we constrain all aforementioned variables to values that are widely used in the literature, or based on our own experiments which have allowed us to select the most appropriate settings. Furthermore, it is noteworthy that we do not aim for absolute performance, but rather measure performance delta between several approaches of constructing training datasets.

2.3 Working Example

In this section, we provide details on the machine-learning approach that will be used as a working example to investigate the importance of history in the selection of training and test datasets. First, we show how our features are sound and how findings based on these can be generalized to state-of-the-art techniques. Second, we briefly describe the features that we use.

A most common feature kind used in the literature of machine-learning is *N-grams* which have shown to be effective for text classification. Kephart has then proposed to used N-grams for malware analysis [28], and was recently followed by a large body of research in malware detection which have leveraged n-grams to generate file/program signatures for the training dataset of malware [24, 29, 37]. We claim however that n-grams, by slicing raw code/data as strings, cannot optimally capture the semantics of most malware, mainly because malicious software increasingly *look like* goodware which they often derive from. For the Android platform, Sahs and Khan have recently proposed [36] to use a combination of Android permission and a representation of a

[1]`http://www.cs.waikato.ac.nz/ml/weka/`

programs' control-flow graphs (CFGs). Unfortunately, not all malware are related to a permission issue, and permissions are often misused by developers [9]. Nevertheless, the CFGs contain much more information on the execution behavior of a program. Thus, we build a technique that extracts, from an application program, data blocks that are semantically more relevant for executed software.

Practically, we perform static analysis of Android applications' bytecode to extract an abstract representation of the program's CFG. We obtain a CFG that is expressed as character strings using a method devised by Pouik *et al.* in their work on establishing similarity between Android applications [34], and that is based on a grammar proposed by Cesare and Xiang [15]. The string representation of a CFG is an abstraction of the application's code; it retains information about the *structure* of the code, but discards low-level details such as variable names or register numbers. This property is desirable in the context of malware detection as two variants of a malware may share the same abstract CFG while having different bytecode. Given an application's abstract CFG, we collect all basic blocks that compose it and refer to them as the features of the application. A basic block is a sequence of instructions in the CFG with only one entry point and one exit point. It thus represents the largest piece of the program that is always executed altogether. By learning from the training dataset, it is possible to expose, if any, the basic blocks that appear statistically more in malware. Overall, using an abstract representation of the code could allow resisting to basic forms of obfuscation, a threat to validity that n-grams-based approaches cannot readily overcome.

The basic block representation used in our approach is a high-level abstraction of small parts of an Android application. Depending on its position inside a method, one sequence of instructions may lead to different bytecode because of a renumbering of VM registers. Our abstract basic block representation however will always produce the same string for one sequence of instructions of a basic block, hence providing a higher resistance to code variations than low-level representations such as n-grams computed on bytecode. For reproducibility purposes, and to allow the research community to build on our experience, the data we used (full feature matrix and labels) is available on request.

2.4 Methodology

This study is carried out as a large scale experiment that aims at investigating the extent of the relevance of history in assessing machine learning-based malware detection. This study is important for paving the road to a true success story of trending approaches to Android malware detection. To this end, our work must rely on an extensive dataset that is representative of real-world Android apps and of datasets used in the state-of-the-art literature. To account for the variety of classification algorithms in machine-learning approaches, we rely on the conclusions of a previous large-scale study by Allix *et al.* [2] with similar machine learning features. Based on this study, we have chosen to use the RandomForrest algorithm which offers high classification performance. To account for other execution parameters such as the ratio between goodware and malware in training datasets, we use standard, or recurrent, settings found in the literature.

Marketplace	# of Android apps	Percentage
`play.google.com`	78 460	38.04%
`appchina`	72 093	34.96%
`1mobile`	32,017	15.52%
`slideme`	17 552	8.51%
`anzhi`	2 141	1.04%
`proandroid`	1 605	0.78%
`fdroid`	1 222	0.59%
`genome`	610	0.3%
`freewarelovers`	369	0.18%
`apk_bang`	168	0.08%

Table 1: Origin of the Android apks in our dataset

Dataset To perform this study we collect a large dataset of android apks from various markets. Table 1 provides information on the distribution of the applications in our dataset following the market where they have been crawled from. A large majority of our dataset comes from `play.google.com`, the official market, and `appchina`.

An Android application is distributed as an `.apk` file which is actually a ZIP archive containing all the resources an application needs to run, such as the application binary code and images. An interesting side-effect of this package format is that all the files that makes an application go from the developers computer to end-users devices without any modification. In particular, all metadata of the files contained in the .apk package, such as the last modification date, are preserved. All bytecode, representing the application binary code, is assembled into a *classes.dex* file that is produced at packaging-time. Thus the last modification date of this file represents the packaging time. In the remainder of this paper, packaging date and compilation date will refer to this date.

To infer the historical distribution of the dataset applications, we rely on compilation date at which the Dalvik[2] bytecode (*classes.dex* file) was produced. We then sent all the app packages to be scanned by virus scanners

[2]Dalvik is the virtual machine running Android apps.

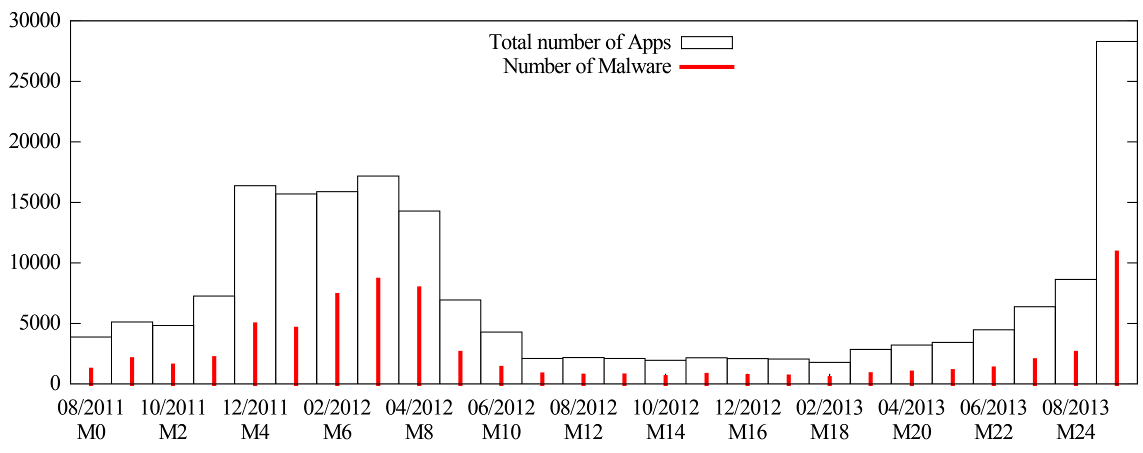


Figure 1: Historical distribution and Malware Proportions

hosted by VirusTotal[3]. VirusTotal is a web portal which hosts about 40 products from renown anti virus vendors, including McAfee®, Symantec® or Avast®. An application is labelled as malware if any scanner flags it as such. Figure 1 depicts the distribution of dataset as well as the percentage of malware found in the subset of each month. The drop in the number of applications in certain months are mainly due to some technical difficulties that we have experienced in the collection of apps from markets.

Machine learning Parameters In all our experiments, we have fixed the number of features to 5,000. We thus selected the 5,000 features with highest Information Gain values as measured on the training sets. The RandomForest algorithm was used for all our experiments.

3 Related Work

In this section, we propose to revisit related work to highlight the benefits of our contributions in this paper. We briefly present previous empirical studies to highlight their significance for the malware detection field. Then we go over the literature of malware detection to discuss their assessment protocol.

3.1 Empirical studies

Empirical studies have seen a growing interest over the years in the field of computer science. The weight of empirical findings indeed help ensure that research directions and results are in line with practices. This is especially important when assessing the performance of a research approach. A large body of the literature has resorted to extensive empirical studies for devising a reliable experimental protocol [10,25,27]. Recently, Allix *et al.* have proposed a large-scale empirical studies on the dataset sizes used in the assessment of machine learning-based malware detection approaches [2]. In their work, the authors already questioned the assessment protocols used in the state-of-the-art literature. Our work follows the same objectives, aiming to highlight the importance of building a reliable assessment protocol for research approaches, in order to make them more useful for real-world problems.

[3]`https://www.virustotal.com`

In the field of computer security, empirical studies present distinct challenges including the scarcity of data about cybercrimes. We refer the reader to a report by Böhme and Moore [11]. Recently, Visaggio *et al.* empirically assessed different methods used in the literature for detecting obfuscated code [41]. Our work is in the same spirit as theirs, since we also compare different methods of selecting training datasets and draw insights for the research community.

With regards to state-of-the-art literature tackled in this work, we note that a significant amount of Machine Learning approaches to malware detection has been presented to the research community [1, 7, 8, 16, 22, 30]. Although most of those approaches could not be reproduced due to undisclosed parameters and/or undisclosed datasets, we have compared their performance indicators with our classifiers in similar 10-Fold evaluation setups. Our classifiers achieved (1) better relative performance metrics, and (2) high absolute performance metrics. This demonstrates the soundness of the feature set we base our experiments on. We further note that in the assessment protocol of all these approaches, the history aspect was eluded when selecting training datasets.

Approach	Year	Sources	Historical Coherence
[36]	2012	Undisclosed	No
DROIDMAT [42]	2012	Contagio mobile for malware, Google Play for goodware	No
[14]	2013	”common Android Markets” for goodware, ”public databases of antivirus companies” for malware	No
[4]	2013	Genome, VirusTotal, Google Play	No
[20]	2013	Contagio mobile and Genome for malware, Undisclosed for goodware	No
[43]	2013	”from official and third party Android markets” for Goodware, Genome for malware	No
[22]	2013	Google Play (labels from 10 commercial Anti virus scanners)	No
DREBIN [7]	2014	”Genome, Google Play, Chinese and russian markets, VirusTotal	No

Table 2: A selection of Android malware detection approaches

3.2 Malware Detection & Assessment of approaches

We now review the assessment of malware detection techniques that are based on machine learning. For comparing performances with our own approach, we focus first on techniques that have been applied to the Android ecosystem. In Table 2, we list recent “successful” approaches from the literature of malware detection. We describe the origin of the dataset used for the assessment of the different approaches. For many of them, the applications are borrowed from known collections of malware samples or from markets such as Google Play. They often use scanners from VirusTotal to construct the ground truth. In our approach, we have obtained our datasets in the same ways. Unfortunately, to the best of our knowledge and according to their protocol descriptions from the literature, none of the authors has considered clearly ordering the data to take into account the history aspect. It is therefore unfortunate that the high performances recorded by these approaches may never affect the fight against malware in applications markets.

Android malware detection The most recent achievement on machine learning-based malware detection for Android is DREBIN [7]. The authors of this approach have relied on different features from the manifest file as well as the byte code to build its SVM classifiers. The performance recorded in their paper are comparable to the performances obtained with our classifiers built during history-unaware 10-Fold experiments. DREBIN goes further by providing explainable outputs, i.e., being able to somehow justify why an app was classified as malware. Unfortunately, these performance records might have the effect of a sword cutting through water since the authors do not mention taking into account real-world constraints such as the relevance of history.

In the remainder of this section we list related work, provide details on the size of their dataset and compare them to our history-unaware 10-Fold experiments. None of them has indeed taken into account the history aspect in their assessment protocol. In 2012, Sahs & Khan [36] built an Android malware detector with features based on a combination of Android-specific permissions and a Control-Flow Graph representation. Their classifier was tested with k-Fold [4] cross validation on a dataset of 91 malware and 2 081 goodware. We obtained comparable values of recall but much higher values for precision and F-measure. Using permissions and API calls as features, Wu et al. [42] performed their experiments on a dataset of 1 500 goodware and 238 malware. Many of our classifiers exhibit higher values of both precision and recall than theirs. In 2013, Amos et al. [4] leveraged dynamic application profiling in their malware detector. The evaluation metrics of their 10-fold experiment are slightly lower than ours. Demme et al. [20] also used dynamic application analysis to perform malware detection with a dataset of 210 goodware and 503 malware. Many of our 10-fold classifiers achieved higher performance than their best classifier. Yerima et al. [43] built malware classifiers based on API calls, external program execution and permissions. Their dataset consists of 1 000 goodware and 1 000 malware. Many of our 10-fold classifiers achieved higher performance than their best classifier. Canfora et al. [14] experimented feature sets based on SysCalls and permissions. Their classifiers, evaluated on a dataset of 200 goodware and 200 malware, yielded lower precision and lower recall than ours.

Windows malware detection We have also explored the research of machine learning-based malware detection for the Windows® platform. Approaches for this platform have indeed inspired Android researchers and experiments on Windows malware have provided insights for dealing with Android malware. As for Android approaches, we describe the features used and the size of datasets. History aspects are however less relevant in these cases, since there is no notion of market and applications apparition/registration time. Kolter & Maloof [29] performed malware classification on Windows Executable files. Using n-grams extracted from those

[4]The value of *k* used by Sahs & Khan was not disclosed.

binary files, and the Information Gain feature selection method, they obtained high performance metrics with 10-Fold experimentations on two collections: The first one consisting in 476 malware and 561 goodware, the second one containing 1 651 malware and 1 971 goodware. Many of our 10-fold classifiers achieved higher performance metrics. In 2006, Henchiri & Japkowicz [24] provided experimental results of a malware detector based on a sophisticated n-grams selection algorithm. They evaluated their classifier using 5-Fold[5] on a dataset of 3 000 samples, of which 1 512 were malware and 1488 were goodware. The majority of our classifiers achieved better results than Henchiri & Japkowicz best ones, even though we used a simple feature selection method. Zhang et al. [44] leveraged a multi-classifier combination to build a malware detector. They evaluated the quality of their detector with the 5-Fold method on three datasets, each containing 150 malware and 423 goodware. The features they are using are based on n-grams, and are selected with InfoGain. Zhang et al. mentions testing on a larger dataset as a future work. Schultz and al. [38] performed malware detection using strings and byte sequences as features. They obtained very high recall and precision with 5-fold Cross Validation on a dataset of 4 266 Windows executables (3 265 known malicious binaries and 1 001 benign). Many of our classifiers performed similarly good or better. Perdisci et al. [31] built a packed executable detector that achieved near 99% accuracy. Their classifiers were trained on 4 493 labelled executables and then tested on 1 005 binaries. The same authors leveraged their packed executable detection method in [32] and added two malicious code detectors, one of which is based on n-grams. They first evaluated one of this detector with 5-fold cross validation on 2 229 goodware and 128 malware and the other detector with 3 856 malware and 169 goodware. Finally, their complete approach called "McBoost" was evaluated with 5-fold on 3 830 malware and 503 goodware. Tahan et al. recently presented "Mal-ID" [40], a malware detector that relies on high-level features obtained with Static Analysis. Their experiments are performed with 10-fold on a dataset built with 2 627 benign executables and 849 known malware.

4 Experimental Findings

In this section, we report on the experiments that we have conducted, and highlight the findings. First we discuss to what extent it is important that datasets remain historically coherent, as opposed to being selected at random (cf. Section 4.1). This discussion is based on qualitative aspects as well as quantitative evaluation. Second, we conduct experiments that attempt to provide a hint to the existence of lineages in Android malware in Section 4.2. Subsequently, we investigate in Section 4.3 the bias in training with new data for testing with old data, and inversely. Finally, we investigate the limitations of a naive approach which would consist in accumulating information on malware samples as time goes, in the hope of being more inclusive in the detection of malware in the future (cf. Section 4.4).

4.1 History-aware Construction of Training datasets

As described in Section 2.2, a key step of machine-learning approaches is the training of classifiers. The construction of the corresponding training dataset is consequently of importance, yet details about how it is achieved are largely missing from the literature.

There are two common selection patterns for training datasets: (1) use a collected and published dataset of malware, such as Genome, to which one adds a subset of confirmed goodware; (2) build the dataset by randomly picking a subset of goodware and malware from a dataset collected from either an online market or an open repository. Both patterns lead to the same situations: i.e. that *some items in the training dataset may be historically posterior to items in the tested dataset*. In other words:

- the construction of the training set is equivalent to a random history-unaware selection from a mix of known malware and goodware
- the history of creation/apparition of android applications is not considered as a parameter in assessment experiments, although the practice of malware detection will face this constraint

Following industry practices, when a newly uploaded set of applications must be analyzed for malware identification, the training datasets that are used are, necessarily, historically anterior to the new set. This constraint is however eluded in the validation of malware detection techniques in the research literature. To clearly highlight the bias introduced by current assessment protocols, we have devised an experiment that compares the performance of the machine learning detectors in different scenarios. The malware detectors are based on classifiers that are built in two distinct settings: either with randomly-constructed training datasets using a process described in Figure 2, or with datasets that respect the history constraint. To reduce the bias between these comparisons, we ensure that the datasets are of identical sizes and with the same class imbalance between goodware and malware. Thus to build a history-unaware dataset R_0

[5]While 10-Fold is equivalent to testing 10 times on 10% while being trained on 90% of the dataset, 5-Fold is equivalent to testing 5 times on 20% while being trained on 80% of the dataset.

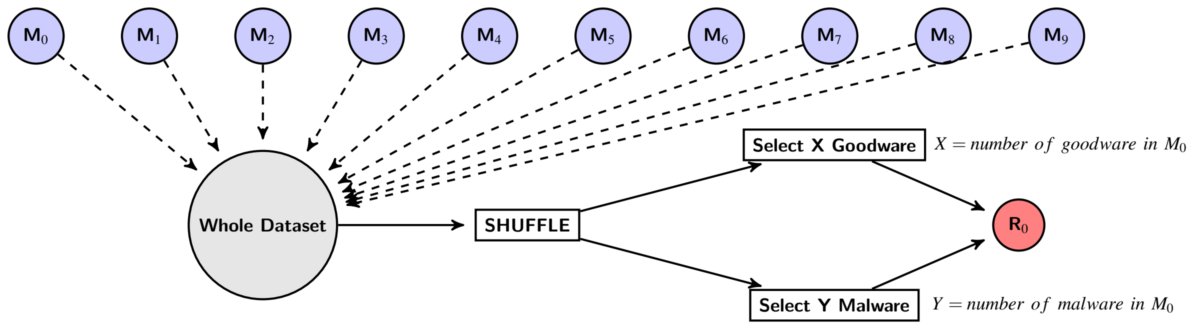


Figure 2: Process of constructing a random training dataset R_0 for comparison with the training dataset constituted of all data from month M_0

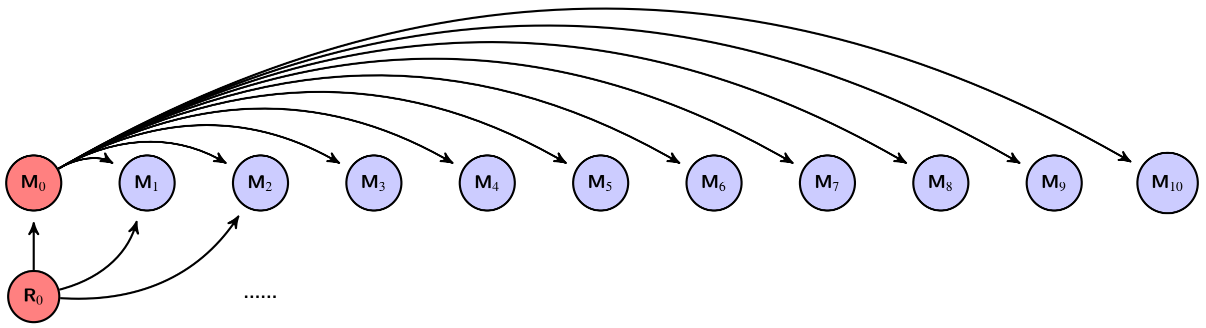
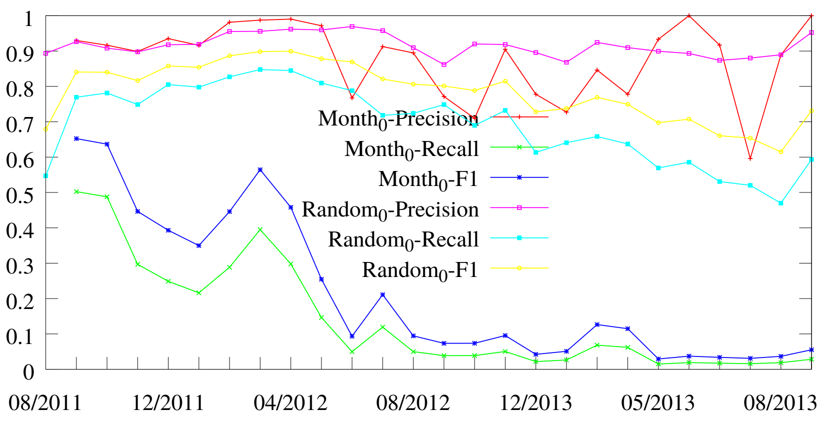


Figure 3: Classification process: the training dataset is either the dataset of a given month (e.g., M_0) or a random dataset constructing as in Figure 2



Reading: The $Month_0$ curves show metrics for a classfier trained on the month 0, while the $Random_0$ curves present results for a classifier built with a training set of same size and same goodware/malware ratio as month 0, but drawn randomly from the whole dataset.

Figure 4: Performance of malware detectors with history-aware and with history-unaware selection of training datasets

for comparing with training dataset constituted of data from month M_0, we randomly pick within the whole dataset the same numbers of goodware and malware as in M_0. We perform the experiments by training first on M_0 and testing on all following months, then by training on R_0 and testing on all months (cf. Figure 3).

Figure 4 illustrates the results of our experiments. When we randomly select the training dataset from the

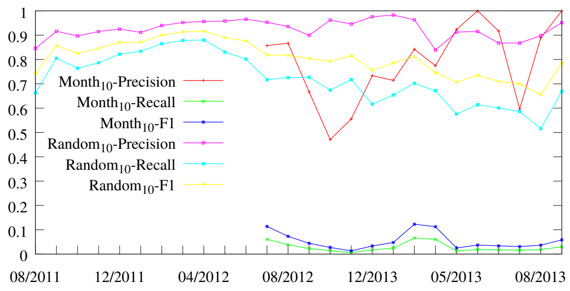


Figure 5: Performance of malware detectors: *the training set is drawn randomly either from* **the whole dataset** *or from* **apps of month** M_{10}

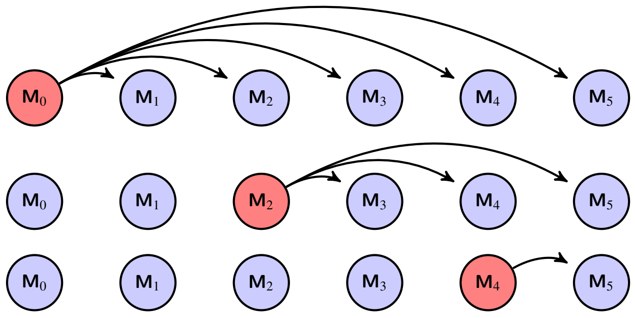


Figure 6: Using training dataset from 1 month for testing on all datasets of following months

entire dataset and build classifiers for testing applications regrouped by month, the precision and recall values of the malware detector range between 0.5 and 0.85. The obtained F-Measure is also relatively high and roughly stable. This performance is in line with the performances of state-of-the-art approaches reported in the literature.

We then proceed to constrain the training dataset to be historically coherent to the test dataset. We select malware and benign apps in the set of apps from a given month, e.g., M_0, as the source of data for building the training dataset for the classification. The tests sets remain the same as in the previous experiments, i.e., the datasets of applications regrouped by month. We observe that as we move away from M_0 to select a test data, the performance considerably drops.

We have repeated this experiment, alternatively selecting each of the different month from our time-line as the month from which we draw the training dataset. Then we only consider testing applications that are posterior to the selected month. Figure 5 illustrates results when selecting training datasets from month M_{10}.

Finding RQ-1: *Constructing a training dataset that is consistent with the history of apparition of applications yields performances that are significantly worst than what is obtained when simply randomly collecting applications in markets and repositories. Thus,* **without further assessment, state-of-the-art approaches cannot be said to be powerful in real-world settings**.

Finding RQ-2: *With random selections, we allow malware "from the future" to be part of the training sets. This however leads to biased results since the performance metrics are artificially improved.*

4.2 Lineages in Android Malware

Our second round of experiments has consisted in investigating the capabilities of a training dataset to help build classifiers that will remain performant over time. In this step of the study we aim at discovering how the variety of malware is distributed across time. To this end, we consider building training datasets with applications in each month and test the yielded classifiers with the data of each following months as depicted in Figure 6.

Figures 7 and 8 provide graphs of the evolution over time of, on the one hand, F-Measure and, on the other hand, Precision and Recall of malware detectors that have been build with a training dataset at month M_i and applied on months $M_{k,k>i}$. Disregarding outliers which lead to the numerous abrupt rise and breaks in the curves, the yielded classifiers have, on average, a stable and high Precision, with values around 0.8. This stability of Precision is confirmed by the shape of the Recall and F-Measure curves. Indeed, these curves is similar, implying that the variations of Recall have more impact on the harmonic measure than the Precision. This finding suggests that *whatever the combination of training and test dataset months, the built classifiers still allow to identify with good precision the malware whose features have been learned during training*.

On the other hand, the Recall performance degrades over time: for a given month M_i whose applications have been used for the training datasets, the obtained classifier is less and less able to identify all malware in the following months $M_{k,k>i}$. This finding, correlated to the previous one, suggests that, over time, the features that are learned in the training dataset correspond less and less to all malware when we are in the presence of **lineages** in the Android malware. We define a `lineage` as a set of malware that share the same traits, whether in terms of behavior or of coding attributes. Note that we differentiate the term `lineage` from the term `family` which, in the literature, concern a set of malware that exploit the

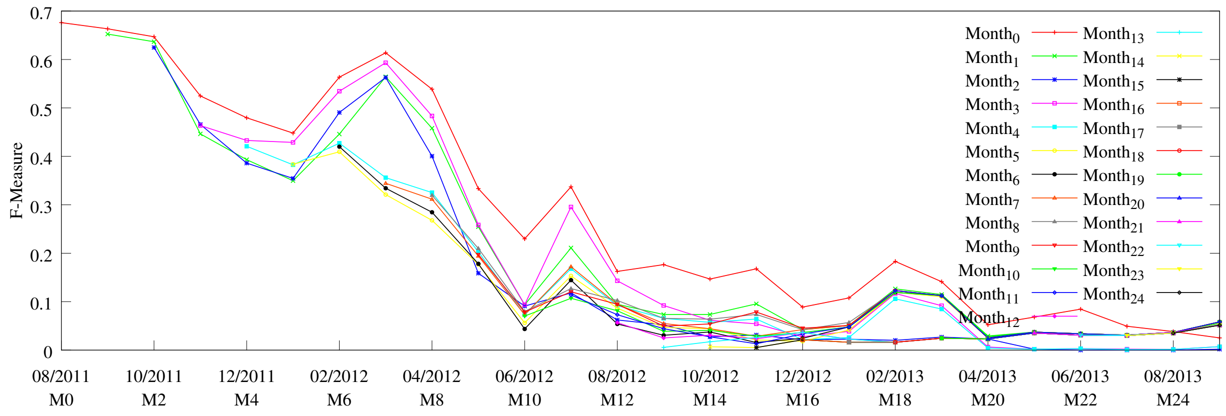
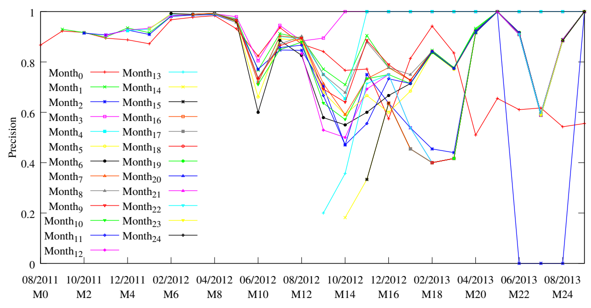


Figure 7: Performance Evolution of malware detectors over time

same vulnerability. *Lineage* is a more general term.

The experiments also highlight the bias introduced when training classifiers with a specific and un-renewed set of malware, such as the Genome dataset. It also confirms why the random selection of malware in the entire time-line as presented in Section 4.1, provides good performances: many lineages are indeed represented in such training datasets, including lineages that should have appeared for the first time in the test dataset.


a) Precision of malware detectors

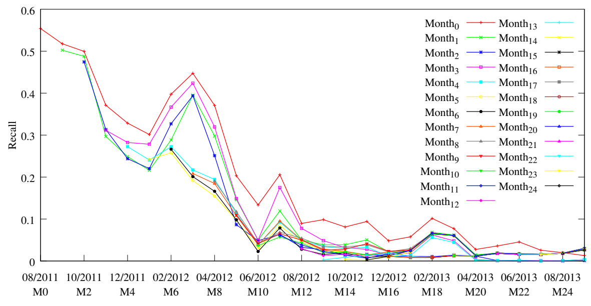

b) Recall of malware detectors

Figure 8: Evolution of Precision and Recall evolution of malware detectors over time

Finding-RQ3: *Android malware is diversified. The existence of lineages complicates malware detection, since training datasets must be regularly updated to include a larger variety of malware lineages representatives.*

4.3 Is knowledge "from the future" the Grail?

Previous experiments have shown that using applications from the entire time-line, without any historical constraint, favorably impacts the performance of malware detectors. We have then proceeded to show that, when the training dataset is too old compared to the test dataset, this performance drops significantly. We now investigate whether training data that are strictly posterior to the test dataset could yield better performance than using data that are historically anterior (coherent). Such a biased construction of datasets is not fair when the objective is to actively keep malicious apps from reaching the public domain. However, such a construction can be justified by the assumption that the present might always contain representatives of malware lineages that have appeared in the past.

In the Android ecosystem, thousands of applications are created weekly by developers. Most of them, including malware from new lineages, cannot be thoroughly checked. Nevertheless, after some time, antivirus vendors may identify the new malware. Machine-learning processes can thus be used to automate a large-scale identification of malware in applications that have been made available for some time. Figure 9 depicts the F-Measure performance evolution of the malware detectors: for each month M_i, that is used for training, the obtained classifiers are used to predict malware in the previous months $M_{k,k<i}$. The Recall evolution, outlined in

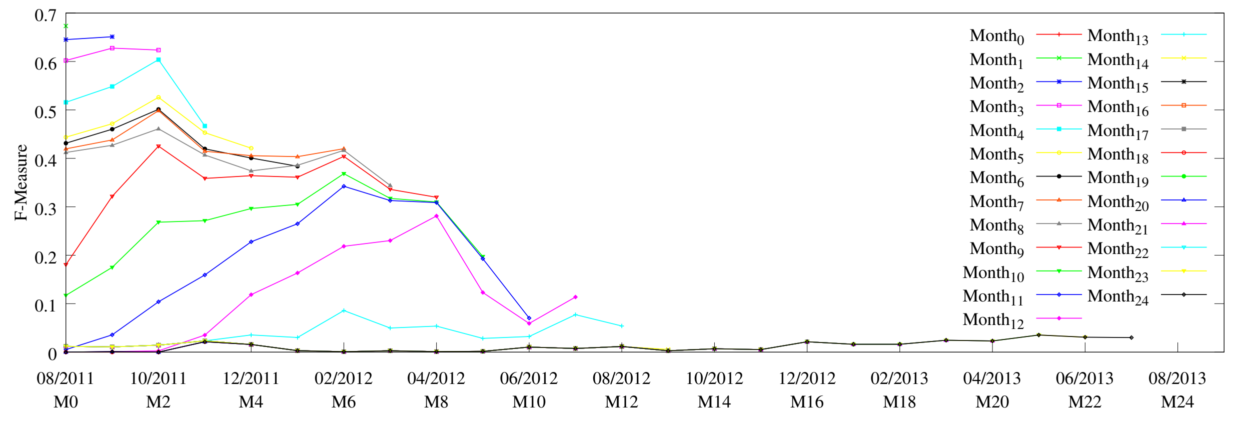
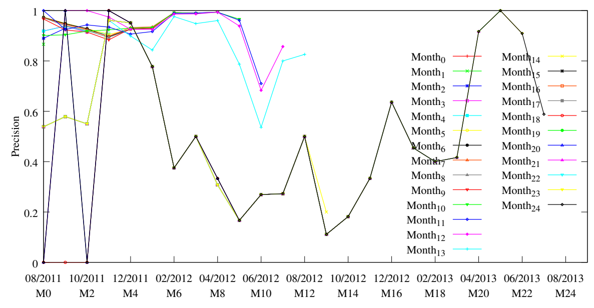
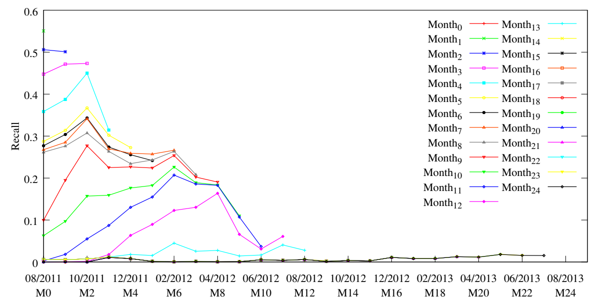


Figure 9: Performance of malware detectors when using recent data to test on old datasets

Figure 10, and the F-Measure evolution follow the same variations, suggesting that the impact of the average variations of the Precision is marginal, despite the shape of the curve which should be attributed to outliers. Overall, the performance is dropping significantly with the time difference between test and training datasets.



a) Precision of malware detectors



b) Recall of malware detectors

Figure 10: Evolution of Precision and Recall of malware detectors when using recent data to test on old datasets

Finding-RQ4: *Applications, including malware, used for training in machine learning-based malware detection must be historically close to the target dataset that is tested. Older training datasets indeed cannot account for all malware lineages, and newer datasets do not contain enough representatives of most malware from the past.*

4.4 Naive Approaches to the Construction of Training Datasets

Given the findings of our study presented in previous sections, we investigate through extensive experiments the design of a potential research approach for malware detection which will be in line with the constraints of industry practices. At a given time t, one can only build classifiers using datasets that are anterior to t. Nevertheless, to improve our chances of maintaining performance, two protocols can be followed:

(1) *Keep renewing the training dataset entirely to stay historically close to the target dataset of test. This renewal process must however be automated to remain realistic.* In this scenario, we assume that a bootstrap step is achieved with antivirus products at month M_0 to provide a first reliable training dataset. The malware detection system is then on its own for the following months. Thus, the classification that is obtained on month M_1, using month M_0 for training, will be used "as is" to train the classifiers for testing applications data of month M_2. This system is iterated until month M_n as depicted in Figure 11, meaning that, once it is bootstrapped, the detection system is automated and only relies on its test results to keep training new classifiers. In practice, such an approach makes sense due to the high precision values recorded in previous experiments.

(2) *Include greedily the most knowledge one can collect on malware lineages* This scenario is also au-

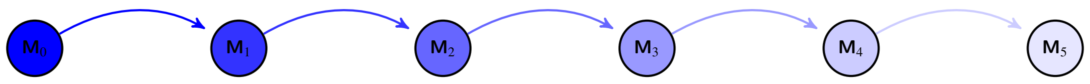
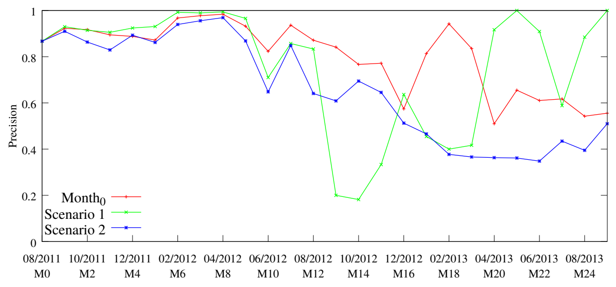
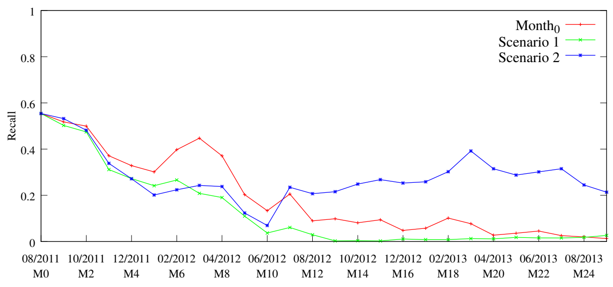


Figure 11: Using malware classification results of month M_{n-1} as training dataset for testing month M_n



a) Comparison based on the Precision metric



a) Comparison based on the Recall metric

Figure 12: Comparing two naive approaches to the baseline of history-aware approach

tomated and requires bootstrapping. However, instead of renewing the training dataset entirely each month, new classification results are added to the existing training dataset and used to build classifiers for the following month.

Figure 12 shows the Precision and Recall performance obtained in both scenarios. We compare these cases to the baseline scenario where we keep using the classifiers obtained at month M_0 with labels from antivirus products to detect malware for every other month (cf. Section 4.1). The graphs show that by constantly using "fresh" data for training in Scenario 1, we maintain the precision at stable and higher rate than the baseline. However, by continuously adding malware to form a training datasets with applications that are more or less fresh, we witness a steady degradation of Precision in Scenario 2. The Recall performance on the other hand follows opposite evolutions. In Scenario 1, using fresh data obtained by classifiers based on the data of previous month, leads to a steady drop in Recall, while adding new malware detected to the training datasets allows to maintain the Recall around to the original baseline.

Both of these naive approaches have opposite advantages and drawbacks. They provide insights, through empirical evidence, on how machine learning-based malware detection systems should consider the construction of training datasets.

Finding-RQ5: *Maintaining performance of malware detectors cannot be achieved by simply adding/renewing information in training datasets based on the output of previous runs. However, these scenarios have shown interesting impact on performance evolution over time, and must be further investigated to identify the right balance.*

5 Discussion, Insights and Future work

In this section we summarize the key points developed in this paper to enumerate the insights provided by our findings. Based on these experimental results, we plan our future work to devise realistic approaches for Android malware detection using machine-learning techniques.

Findings

- History constraints must not be eluded in the construction of training datasets of machine learning-based malware detectors. Indeed, they introduce significant bias in the interpretation of the performance of malware classifiers.

- There is a need for building a reliable, and continuously updated, benchmark for machine learning-based malware detection approaches. We make available, upon request, the version we have built for this work. Our benchmark dataset contains about 200,000 Android applications spanning 2 years of historical data of Android malware.

Insights

- Machine-learning cannot assure the identification of an entirely new lineage of malware that is not represented in the training dataset. Thus, there is need to regularly seed the process with outside information, such as from antivirus vendors, of new lineages of malware.

- In real world settings, practitioners cannot be presented with a reliable dataset for training. Indeed, most malware are discovered, often manually, by antivirus vendors far later after they have been available to end-users. Large-scale ML-based malware detection must therefore be used to automate the discovery of variants of malware which have been authenticated in a separate process.

Finding-RQ6: *Building a reliable training dataset for obtaining the best real-world performance requires a right balance between the need for maximizing fresh data and the need for including samples of most malware lineages.*

Threat to Validity

To perform this study, we have considered a unique use-case scenario for using machine learning-based malware detection. This scenario consists in *Actively preventing malware from reaching markets* and is extremely relevant to most real-world constraints. Indeed, in practice, it is important to keep the window of opportunity very narrow. Thus, to limit the number of infected devices, Android malware must be detected as they arrive in the market. It is therefore important that state-of-the-art approaches be properly assessed, taking into account history constraints.

There is however a second use-case scenario which concerns online repositories for research and would consist on *cleaning such repositories regularly*. In this scenario, repositories maintainers attempt to filter malicious apps once a new kind of malware has been discovered. In such a context, practitioners can afford to wait for a long time before building relevant classifiers for identifying malware that have been since in the repository. Nevertheless, such repositories are generally of reasonable size and can be scanned manually and with the help of anti virus products.

Future work

- Building on the insights of our experiments, we plan to investigate how to maintain the performance of machine learning-based malware detectors until antivirus vendors can detect a new strain of malware. This research direction could help bring research work to be applied on real-world processes, in conjunction with antivirus products which are still widely used, although they do not scale to the current rates of malware production.
- To account for the evolution of representations of malware lineages in training datasets over time, we plan to investigate a multi-classifier approach, each classifier being trained with more or less outdated data and weighted accordingly. A first challenge will be on how to infer or automate the choice of weights for different months in the timeline to build the most representative training dataset.

6 Conclusion

Given the steady increase in the adoption of smartphones worldwide, and the growth of application development for such devices, it is becoming important to protect users from the damages of malicious apps. Malware detection has thus been in recent years the subject of renewed interest, and researchers are investigating scalable techniques to spot and filter out apps with malicious traits among thousands of benign apps.

However, more than in other fields, research in computer security must yield techniques and approaches that are truly usable in real-world settings. To that end, assessment protocols of malware detection research approaches must reflect the practice and constraints observed by market maintainers and users. Through this empirical study we aim to prevent security research from producing approaches and techniques that are not in line with reality. Furthermore, given the performances reported in state-of-the-art literature of malware detection, while market maintainers still struggle to keep malware out of markets, it becomes important to clear the research field by questioning current assessment protocols.

In this paper, we have investigated the relevance of history in the selection of assessment datasets. We have performed large-scale experiments to highlight the different bias that can be exhibited by different scenarios of dataset selection. Our main conclusion is that the assessment protocol used for current approaches in the state-of-the-art literature is far from the reality of a malware detection practice for keeping application markets clean. We have further investigated naive approaches to training dataset construction and drawn insights for future work by the research community.

References

[1] AAFER, Y., DU, W., AND YIN, H. Droidapiminer: Mining api-level features for robust malware detection in android. In *Proceedings of the International Conference on Security and Privacy in Communication Networks* (2013), SecureComm.

[2] ALLIX, K., BISSYANDÉ, T. F., JEROME, Q., KLEIN, J., STATE, R., AND LE TRAON, Y. Large-scale machine learning-based malware detection: Confronting the ”10-fold cross validation”scheme with reality. In *Proceedings of the Fifth ACM Con-*

ference on Data and Application Security and Privacy (2014), CODASPY.

[3] ALPAYDIN, E. *Introduction to Machine Learning*, 2nd ed. The MIT Press, 2010.

[4] AMOS, B., TURNER, H., AND WHITE, J. Applying machine learning classifiers to dynamic android malware detection at scale. In *Proceedings of 9th International Wireless Communications and Mobile Computing Conference* (2013), IWCMC, pp. 1666–1671.

[5] APPBRAIN. Number of available android applications. `http://www.appbrain.com/stats/number-of-android-apps`. Accessed: 2013-09-09.

[6] APVRILLE, A., AND STRAZZERE, T. Reducing the window of opportunity for android malware gotta catch 'em all. *Journal of Computer Virology 8*, 1-2 (May 2012), 61–71.

[7] ARP, D., SPREITZENBARTH, M., HÜBNER, M., GASCON, H., AND RIECK, K. Drebin: Effective and explainable detection of android malware in your pocket. In *Proceedings of the Network and Distributed System Security Symposium* (2014), NDSS.

[8] BARRERA, D., KAYACIK, H., VAN OORSCHOT, P., AND SOMAYAJI, A. A methodology for empirical analysis of permission-based security models and its applications to android. In *Proceedings of ACM Conference on Computer and Communications Security* (2010), CCS.

[9] BARTEL, A., KLEIN, J., LE TRAON, Y., AND MONPERRUS, M. Automatically securing permission-based software by reducing the attack surface: an application to android. In *Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on* (Sept 2012), pp. 274–277.

[10] BISSYANDÉ, T. F., THUNG, F., WANG, S., LO, D., JIANG, L., AND RÉVEILLÈRE, L. Empirical Evaluation of Bug Linking. In *17th European Conference on Software Maintenance and Reengineering (CSMR 2013)* (Genova, Italy, Mar. 2013), pp. 1–10.

[11] BÖHME, R., AND MOORE, T. Challenges in empirical security research. Tech. rep., Singapoore Management University, 2012.

[12] BOSHMAF, Y., RIPEANU, M., BEZNOSOV, K., ZEEUWEN, K., CORNELL, D., AND SAMOSSEIKO, D. Augur: Aiding malware detection using large-scale machine learning. In *Proceedings of the 21st Usenix Security Symposium (Poster session)* (Aug 2012).

[13] BREIMAN, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[14] CANFORA, G., MERCALDO, F., AND VISAGGIO, C. A. A classifier of malicious android applications. In *Proceedings on the 8th Conference on Availability, Reliability and Security (ARES)* (2013).

[15] CESARE, S., AND XIANG, Y. Classification of malware using structured control flow. In *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing - Volume 107* (Darlinghurst, Australia, Australia, 2010), AusPDC '10, Australian Computer Society, Inc., pp. 61–70.

[16] CHAKRADEO, S., REAVES, B., TRAYNOR, P., AND ENCK, W. Mast: Triage for market-scale mobile malware analysis. In *Proceedings of ACM Conference on Security and Privacy in Wireless and Mobile Networks* (2013), WISEC.

[17] CHAU, D. H., NACHENBERG, C., WILHELM, J., WRIGHT, A., AND FALOUTSOS, C. Polonium: Tera-scale graph mining for malware detection. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (2010).

[18] COHEN, W. W. Fast effective rule induction. In *Proceedings of the International Machine Learning Conference* (1995), Morgan Kaufmann Publishers, Inc., pp. 115–123.

[19] CORTES, C., AND VAPNIK, V. Support-vector networks. *Machine Learning 20*, 3 (1995), 273–297.

[20] DEMME, J., MAYCOCK, M., SCHMITZ, J., TANG, A., WAKSMAN, A., SETHUMADHAVAN, S., AND STOLFO, S. On the feasibility of online malware detection with performance counters. In *Proceedings of the 40th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2013), ISCA '13, ACM, pp. 559–570.

[21] ENCK, W., OCTEAU, D., MCDANIEL, P., AND CHAUDHURI, S. A study of android application security. In *Proceedings of the 20th USENIX conference on Security* (San Francisco, CA, 2011), SEC'11, pp. 21–21.

[22] GASCON, H., YAMAGUCHI, F., ARP, D., AND RIECK, K. Structural detection of android malware using embedded call graphs. AISec.

[23] GOOGLE. Android and security (bouncer announcement). `http://googlemobile.blogspot.fr/2012/02/android-and-security.html`, Feb. 2012. Accessed: 2013-05-30.

[24] HENCHIRI, O., AND JAPKOWICZ, N. A feature selection and evaluation scheme for computer virus detection. In *Proceedings of the Sixth International Conference on Data Mining* (Washington, DC, USA, 2006), ICDM '06.

[25] HUTCHINS, M., FOSTER, H., GORADIA, T., AND OSTRAND, T. Experiments of the effectiveness of dataflow-and controlflow-based test adequacy criteria. In *Proceedings of the 16th international conference on Software engineering* (1994), ICSE, pp. 191–200.

[26] IDIKA, M. A survey of malware detection techniques. Tech. rep., Purdue University, Feb. 2007.

[27] JONES, J. A., AND HARROLD, M. J. Empirical evaluation of the tarantula automatic fault-localization technique. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering* (2005), ASE, ACM, pp. 273–282.

[28] KEPHART, J. O. A biologically inspired immune system for computers. In *Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems* (1994), Artificial Life IV, MIT Press, pp. 130–139.

[29] KOLTER, J. Z., AND MALOOF, M. A. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research 7* (Dec. 2006), 2721–2744.

[30] PENG, H., GATES, C. S., SARMA, B. P., LI, N., QI, Y., POTHARAJU, R., NITA-ROTARU, C., AND MOLLOY, I. Using probabilistic generative models for rangking risks of android apps. In *Proceedings of ACM Conference on Computer and Communications Security* (2012), CCS.

[31] PERDISCI, R., LANZI, A., AND LEE, W. Classification of packed executables for accurate computer virus detection. *Pattern Recognition Letters 29*, 14 (2008), 1941 – 1946.

[32] PERDISCI, R., LANZI, A., AND LEE, W. Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In *Proceedings of the Annual Computer Security Applications Conference* (2008), ACSAC, pp. 301–310.

[33] PIETERSE, H., AND OLIVIER, M. Android botnets on the rise: Trends and characteristics. In *Proceedings of the Conference on Information Security for South Africa* (2012), ISSA, pp. 1–5.

[34] POUIK, AND G0RFI3LD. Similarities for fun & profit. *Phrack 14*, 68 (April 2012). `http://www.phrack.org/issues.html?id=15&issue=68`.

[35] QUINLAN, J. R. *C4. 5: programs for machine learning*, vol. 1. Morgan kaufmann, 1993.

[36] SAHS, J., AND KHAN, L. A machine learning approach to android malware detection. In *Proceedings of the IEEE European Intelligence and Security Informatics Conference* (2012), EISIC, pp. 141–147.

[37] SANTOS, I., PENYA, Y. K., DEVESA, J., AND BRINGAS, P. G. N-grams-based file signatures for malware detection. In *Proceedings of the International Conference on Enterprise Information Systems* (2009), ICEIS, pp. 317–320.

[38] SCHULTZ, M., ESKIN, E., ZADOK, E., AND STOLFO, S. Data mining methods for detection of new malicious executables. In *IEEE Symposium on Security and Privacy* (2001), S&P, pp. 38–49.

[39] SU, X., CHUAH, M., AND TAN, G. Smartphone dual defense protection framework: Detecting malicious applications in android markets. In *Eighth IEEE International Conference on Mobile Ad-hoc and Sensor Networks* (2012), MSN, pp. 153–160.

[40] TAHAN, G., ROKACH, L., AND SHAHAR, Y. Mal-id: Automatic malware detection using common segment analysis and meta-features. *Journal of Machine Learning Research 13* (Apr. 2012), 949–979.

[41] VISAGGIO, C. A., PAGIN, G. A., AND CANFORA, G. An empirical study of metric-based methods to detect obfuscated code. *International Journal of Security & Its Applications 7*, 2 (2013).

[42] WU, D.-J., MAO, C.-H., WEI, T.-E., LEE, H.-M., AND WU, K.-P. Droidmat: Android malware detection through manifest and api calls tracing. In *Proceedings of the 7th Asia Joint Conference on Information Security* (2012), AsiaJCIS, pp. 62–69.

[43] Yerima, S., Sezer, S., McWilliams, G., and Muttik, I. A new android malware detection approach using bayesian classification. In *Proceedings of the 27th IEEE International Conference on Advanced Information Networking and Applications* (2013), AINA, pp. 121–128.

[44] Zhang, B., Yin, J., Hao, J., Zhang, D., and Wang, S. Malicious codes detection based on ensemble learning. In *Proceedings of the 4th international conference on Autonomic and Trusted Computing* (Hong Kong, China, 2007), ATC, pp. 468–477.