Empirical Software Engineering manuscript No.
(will be inserted by the editor)

Empirical Assessment of Machine Learning-Based Malware Detectors for Android

Measuring the Gap between In-the-Lab and In-the-Wild Validation Scenarios

Kevin Allix · Tegawendé F. Bissyandé · Quentin Jérome · Jacques Klein · Radu State · Yves Le Traon

Abstract To address the issue of malware detection through large sets of applications, researchers have recently started to investigate the capabilities of machine-learning techniques for proposing effective approaches. So far, several promising results were recorded in the literature, many approaches being assessed with what we call *in the lab* validation scenarios. This paper revisits the purpose of malware detection to discuss whether such *in the lab* validation scenarios provide reliable indications on the performance of malware detectors in real-world settings, aka *in the wild*.

To this end, we have devised several Machine Learning classifiers that rely on a set of features built from applications' CFGs. We use a sizeable dataset of over 50 000 Android applications collected from sources where state-of-the art approaches have selected their data. We show that, *in the lab*, our approach outperforms existing machine learning-based approaches. However, this high performance does not translate in high performance *in the wild*. The performance gap we observed—F-measures dropping from over 0.9 *in the lab* to below 0.1 *in the wild*—raises one important question: How do state-of-the-art approaches perform *in the wild*?

Keywords Machine Learning, Ten-Fold, Malware, Android

1 Introduction

The momentum of malware detection research is growing, stimulated by the rapid spread of mobile malware. Indeed, the increasing adoption of smartphones and electronic tablets has created unprecedented opportunities of damages by malicious software which are hidden among the millions of mobile apps available, often for free, on application markets (Felt et al 2011). This reality is currently witnessed on the Android platform, where more and more users of Android-enabled smartphones and other handheld devices are able to install third party applications from both official

K. Allix · T. F. Bissyandé · Q. Jérome · J. Klein · R. State · Y. Le Traon
Interdisciplinary Center for Security, Reliability and Trust, University of Luxembourg, 4 rue Alphonse Weicker, L-2721 Luxembourg, Luxembourg
E-mail: {firstname.lastname}@uni.lu

and alternative markets. In such a context, the security of devices as well as the security of the underlying network have become an essential challenge for both the end users and their service providers. Malware pose various threats that range from simple user tracking and leakage of personal information (Enck et al 2011), to unwarranted premium-rate subscription of SMS services, advanced fraud, and even damaging participation to botnets (Pieterse and Olivier 2012). Although these threats are equally important in both the desktop computing world and the mobile computing world, most users of handheld devices fail to realize the severity of the dangers these devices expose them to. This situation is further exacerbated by the fact that Antivirus vendors have not yet achieved the same kind of performance that they have achieved for personal computers, nor will they be given the time to do so by developers of mobile malware.

Machine learning techniques, by allowing to sift through large sets of applications to detect malicious applications based on measures of similarity of features, appear to be promising for large-scale malware detection (Henchiri and Japkowicz 2006; Kolter and Maloof 2006; Zhang et al 2007; Sahs and Khan 2012; Perdisci et al 2008b). Unfortunately, measuring the quality of a malware detection scheme has always been a challenge, especially in the case of malware detectors whose authors claim that they work "in the wild". Furthermore, when the approach is based on machine learning, authors often perform a 10-Fold cross validation experiment on small datasets to assess the efficiency of the approach. This combination of 10-Fold Cross Validation and small dataset is what we call an *in the lab* scenario. However, we claim that, in the field of malware detection, all the underlying hypotheses associated with an *in the lab* experiment must be outlined to allow a correct interpretation of the results. Indeed, validation experiments of malware detection approaches are often controlled and the datasets used may not be representative, both in terms of *size* and in terms of *quality*, of the targeted universe.

The present paper is both an illustration and a complement to the study published by Rossow et al (2012) and called "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook". Our work focuses on realistic empirical assessment, one of the many issues raised by Rossow et al. In their introduction, they state:

> [...] we find that published work frequently lacks sufficient consideration of experimental design and empirical assessment to enable translation from proposed methodologies to viable, practical solutions. In the worst case, papers can validate techniques with experimental results that suggest the authors have solved a given problem, but the solution will prove inadequate in real use.

Indeed, while most of the studies presented in our related work section (7) were published *after* the paper of Rossow et al., they all present this very shortcoming in their validation methodology.

This paper. We discuss in this paper a new machine learning-based malware detection approach that is effective when assessed with the *in the lab* validation scenario. However, our work aims at shedding light on whether *a high performance recorded with a typical* in the lab *experiment guarantees even a good performance in realistic malware detection use-cases*. To this end, we proceed to compare the performance of machine learning classifiers when they are being validated *in the lab* and when they are used in the wild (i.e., the way they are intended to be used). Due to the scarcity of author data and the lack of sufficient implementation details to reproduce approaches from the state-of-the art literature, we base our investigation on our newly designed malware detection approach. We have devised several machine learning classifiers and built a

set of features which are textual representations of basic blocks extracted from the Control-Flow Graph of applications' bytecode. We use a sizeable dataset of over 50 000 Android applications collected from sources that are used by authors of state-of-the art approaches.

The contributions of this paper are:

- We propose a feature set for machine-learning classifiers for malware detection.
- We show that our implemented classifiers yield a high malware discriminating power when evaluated and compared with state of the art techniques from the literature. This *in the lab* evaluation is based on the 10-Fold cross validation scheme which is popular in the machine learning-based malware detection community.
- We demonstrate limitations of this validation scenario that is performed in the literature of malware detection. In particular, we show with abundant experimental data that 10-Fold validation on the usual sizes of datasets presented in the literature is not a reliable performance indicator for realistic malware detectors.

This paper is organised as follows. Section 2 discusses malware detection in the wild and highlights the associated challenges. We provide in Section 3 various information on the datasets of our experiments, the investigated research questions as well as the used evaluation metrics. Section 4 describes our approach of malware detection, exploring the variables that can be parameterized to tune the output of the machine learning process. Section 5 presents the assessment of our approach, highlighting its performance against state of the art approaches, but also showing its counter-performance in the wild. Section 6 discusses potential threats to validity. Related work is discussed in Section 7. Section 8 concludes and enumerates future work.

2 Malware Detection in the Wild

The market share of Android and its open source architecture has made it a primary target for malware attacks among mobile operating systems. In the official Android application store, *Google Play*, up to 40 000 new applications are registered in a month according to AppBrain (2013b). In this context, especially for alternative markets, it is important to devise malware detection approaches that are efficient in: (1) quickly identifying, with *high precision*, new malware among thousands of newly arrived applications, (2) classifying a large set of applications to expose its *entire* subset of suspicious ones.

Machine learning is a tool used in Artificial Intelligence to provide computers with capabilities for automatically improving themselves in the recognition of patterns. Machine-learning algorithms rely on selected features and training data to infer the commonalities that a group of searched items share and that discriminate them from the rest of the universe. The success of these algorithms therefore depend on the *relevance of the features* for discrimating between the group of searched items and the rest, and on *the quality of training data* for being unbiased and representative of the universe of items. In machine learning-based malware detection, there is a challenge to meet both requirements. Indeed, in the wild, i.e., in real-world scenarios, there are much more goodware than malware, and it is yet difficult to build a set of "perfect" goodware that does not contain a single malware. Consequently, validation of the performance of malware detectors should reflect these specificities. Indeed:

- Using small datasets of goodware and malware of similar size cannot guarantee a realistic assessment of a malware detector that is intended to be used in the wild.
- Blindly using a goodware set without properly validating that it does not contain malware will significantly bias the yielded results

3 Data Sources, Research Questions and Metrics

In this section, we mainly present the datasets that are used to assess our malware detection approach as well as the different aspects that are evaluated.

3.1 Datasets

For our experiments we have used two sources of Android applications that are often used by researchers and practitioners of machine learning-based malware detection for Android. However, to the best of our knowledge our dataset is the largest ever presented in the Android malware detection literature. We make it available to the research community.

Building an Android market dataset. Google Play[1] is the main Android applications market available, and thus constitutes a unique source of relevant applications that are used and that reflects the state of Android application development. We have built a tool that automatically crawls and downloads free applications available in this source. Due to limitations in the implementation of our tool and to restrictions set by Google regarding automatic crawling, we could not retrieve all free applications. Nonetheless, in the course of six (6) months, we have collected a sizeable dataset of nearly 52 000 unique applications. Although Google use various tools to keep Google Play free of malware, we found, after investigation with antivirus, that our collected dataset includes malware.

Collecting known malware. For training needs, we must have access to a reliable and representative set of Android malware. To this end, we leverage a dataset released in the course of the Genome project by researchers from the North Carolina State University (Zhou and Jiang 2012). The *Genome dataset* contains over 1 200 Android malware samples.

3.2 Research Questions & Metrics

We now discuss four important research questions that we have formulated to assess the effectiveness of our machine learning-based malware detectors.

RQ1. *What is the sensitivity of the malware detector when the Goodware/Malware ratio changes in training data?* Because training data is an important element of a machine learning process, we investigate the impact of the composition of this data on the output of the malware detector.

[1] Google Play was formerly known as *Google Market*

RQ2. *How does the number of selected features influence the performance of the tool?* We study the correlation between the number of features used to discriminate malware and the performance of the malware detection scheme.

RQ3. *What is the impact of the underlying machine learning algorithm?* With this research question we want to assess that the algorithm that is used for the implementation of our approach does not significantly bias our findings.

RQ4. *What is the sensitivity of the tool towards the quality of training data?* In the wild, the supposed goodware dataset may be imperfect and contain unknown malware, hence adding noise to the training phase. We investigate the impact that such misrepresentations in training data can have to the final output of the malware detector.

Those four research questions contribute to the common goal of determining the performance of a malware detector for several sets of parameters. Indeed, evaluating a malware detector for one fixed set of parameters only tells the experimenter how it would perform under the exact same conditions.

Malware labeling. For the purpose of guaranteeing a reliable assessment of our approach, we undertake to label all applications by classifying them beforehand as malware or goodware, thus building the ground truth. To construct a reference independent classification to which we can compare the predictions yielded by our machine learning-based approach, we collected from VirusTotal[2] the analysis report of each application in our datasets. VirusTotal is a service that allows security practitioners to readily obtain information on antivirus products which have identified a given application sample as malware. At the time of writing, VirusTotal supported around 40 different antivirus products which are continuously updated both in terms of software release version and in terms of malware databases. Several thousands of the malware in our datasets were unknown to VirusTotal before we submitted them.

Assessment metrics. To quantitatively evaluate the efficacy of our approach, we propose to use standard metrics from the field of Information Retrieval, namely the Precision, Recall, and F-measure metrics.

- **Precision**, as captured by Equation (1), quantifies the effectiveness of the tool to identify suspicious applications that are actually malware. When the tool reports applications as malware and all turn out to be as such, its Precision amounts to 1.

$$Precision = \frac{|\{labeled\ malware\} \cap \{malware\ inferred\ by\ tool\}|}{|\{malware\ inferred\ by\ tool\}|} \quad (1)$$

- **Recall** on the other hand explores the capability of the tool to identify most of the malware. Equation (2) provides the formula for its computation. A Recall evaluated to 0 indicates that no actual malware in the test set has been identified as such by the tool.

$$Recall = \frac{|\{labeled\ malware\} \cap \{malware\ inferred\ by\ tool\}|}{|\{labeled\ malware\}|} \quad (2)$$

[2] https://www.virustotal.com

– Finally, we compute the **F-Measure**, the harmonic mean between Recall and Precision. We consider that both Precision and Recall are equally important and thus, they are equally weighted in the computation of F-measure in Equation (3).

$$F\text{-}Measure = F_1 = 2 \cdot \frac{Precision \times Recall}{Precision + Recall} \tag{3}$$

4 Experimental Setup

Malware detection shares a few challenges with other field of computer science such as natural language processing where information retrieval techniques can be leveraged to isolate and retrieve information that is hard to see at first glance. For text classification (Jacob and Gokhale 2007), researchers often rely on approaches based on n-grams, which, given a string of length M, are all the substrings of length n ($with\ n < M$) of this string. The difficulty in malware detection consists in recognizing, for classification purpose, the signature of a malware. Already in 1994, Kephart at IBM has proposed to use N-grams for malware analysis (Kephart 1994). More recently a large body of research in malware detection based on machine learning have opted for n-grams to generate file/program signatures for the training dataset of malware (Henchiri and Japkowicz 2006; Kolter and Maloof 2006; Santos et al 2009). Despite the high performance claimed by the authors for very small datasets, between 500 and 3 000 software programs, we believe that a malware detector based on n-grams, because of its vulnerability to obfuscation, could be trivially defeated by malware authors. For the Android platform, Sahs and Khan (2012) recently proposed to use a combination of Android permission and a representation of programs' control-flow graphs. However, since all malware are not related to a permission issue, we believe that their approach will yield poor results for other various types of malware.

In this paper we propose a different approach to extract, from an application program, data blocks that are semantically more relevant for executed software. These blocks are elements of applications' Control Flow Graphs which should capture, in a more meaningful way than n-grams, the implementation of a malicious behavior inside program code.

4.1 Our Feature Set for Malware Detection

As detailed in previous sections, machine learning-based malware detection relies on a training data that is analyzed to *learn what could suggest that a given application is a potential malware*. To that end, the learning algorithm must be "told" what features are relevant in each piece of data of the dataset. Indeed, Machine Learning algorithms cannot work directly on Android applications; Each application must be represented with an ordered list of properties—called a Feature vector in the context of Machine Learning. Several sets of features designed to characterize executable code have been introduced in previous approaches (Cf. section 7).

Features are often extracted from program metadata or program code (binaries, bytecode, source code). In the case of the Android Operating System, features can be extracted from application bytecode using static analysis. Indeed, Android applications are distributed in the form of `.apk` files which are packages containing the application's

Dalvik[3] bytecode, assets such as images, and metadata specific to the Android platform. Android applications are generally written in Java. The program is then compiled to Java bytecode which is converted into Dalvik bytecode. Unlike the typical binary code, Dalvik bytecode retains most of the information contained in Java bytecode. Thus, such code can be fed to Static Analysis tools that support Dalvik bytecode or after converting it back to Java Bytecode for which many analyzers exist. In our work, the static analysis was performed using AndroGuard.

We perform static analysis of Android applications' bytecode to extract a representation of the program control-flow graph (CFG). The extracted CFG is expressed as character strings using a method devised by Pouik *et al.* in their work on establishing similarity between Android applications (Pouik et al. 2012). This method is based on a grammar proposed by Cesare and Xiang (2010). This derived string representation of the CFG is an abstraction of the application's code that retains information about the *structure* of the code, but discards low-level details such as variable names or register numbers. In the context of malware detection, this is a desirable property. Indeed, two variants of a malware may share the same abstract CFG while having different bytecode. Thus, using an abstract representation of the code could allow to resist to basic forms of obfuscation, a threat to validity that n-grams-based approaches cannot readily overcome.

Given the abstract representation of an application's CFG, we collect all basic blocks that compose 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 smallest 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.

Let us note BB_i a basic block and BB_{all} the set of the n basic blocks encountered at least in one application.

$$BB_{all} = \{BB_1, BB_2, \cdots, BB_n\} \tag{4}$$

For every application *App*, we build a list, $Features_{App}$, of binary values $(0, 1)$ that codifies all basic blocks from BB_{all} that appear in the *App* and those that do not.

$$Features_{App} = (b_{App,1}, b_{App,2}, \cdots, b_{App,n}) \tag{5}$$

In Equation 5, $b_{App,i}$ is set to 1 if the basic block BB_i is present in the abstract CFG of *App*, and 0 otherwise.

Experimental analysis with all applications from our datasets have shown that with this method, we could extract over 2.5 millions different basic blocks, each appearing once or more in the CFGs of applications. 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 register renumbering. 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 feature matrices that we have computed for both the Genome and the Google Play dataset are publicly available for download[4].

[3] Dalvik is a virtual machine that is included in the Android OS

[4] https://github.com/malwaredetector/malware-detect

4.2 Classification Model

Classification in machine learning-based approaches is the central phase during which an algorithm assigns items in a collection to target classes. In our case, the classification phase aims at predicting if a given application should be assigned to the malware class. In preparation to the classification phase, we must build a dataset in which the class assignments, i.e., goodware or malware, are known for the application. The classification model is then built by a classification algorithm which attempts to find relationships between the features of the applications and their class assignments. This process is known as the *training* phase of the algorithm. In our approach we rely on four (4) well-known classification algorithms, namely Support Vector Machine (SVM) (Cortes and Vapnik 1995), the RandomForest ensemble decision-trees algorithm (Breiman 2001), the RIPPER rule-learning algorithm (Cohen 1995) and the tree-based *C4.5* algorithm (Quinlan 1993).

We now discuss the different steps, illustrated in Figure 1, for building the classification model.

Step 0: Set composition Our complete dataset contains over 50 000 applications that we divide into two distinct sets, one significantly smaller than the other, for the purpose of assessment. The first set, Set_α, contains all known malware, i.e., all items in the *Genome* dataset. To complete this set, we randomly select a subset of the *Google Play* dataset to add as the goodware portion of the dataset. The second set, Set_δ, is then composed of the remaining subset of the Google Play dataset. Set_δ is always used as a testing set, whereas Set_α can be used as training set (in the wild) or as the entire universe (10-Fold), i.e., testing and training sets combined (cf. Fig. 1).

Step 1: Feature Evaluation Once the sets of an experiment are defined, a feature evaluation step is performed to measure the discriminating power of every feature. This measure is computed using the InfoGain Feature evaluation as implemented in the Machine Learning software Weka[5] (Hall et al 2009).

Step 2: Feature Selection For practical reasons, given the large sizes of the datasets, hence the high number of features to process, we must improve computation efficiency by reducing the number of features. Indeed, reducing the number of features considered for the classification will decrease the working size of the sets, leading to lowered I/O, memory and CPU consumption for the subsequent processing steps. In our approach we only retain, after the evaluation step, the best N features, i.e. those with the highest InfoGain values. The number of features is reduced in both the training set and the testing set. For every built training set, we derived about 2.5 millions features, and over 99% of them had a null (0) InfoGain measure. We thus discard those features whose null discrimination power implies that they are "irrelevant". Previous work has already demonstrated that removing such irrelevant features may, beyond computation efficiency gain, improve classifiers' ability to generalize its model (Tahan et al 2012), which in turn could lead to a better detection of previously unknown malware.

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

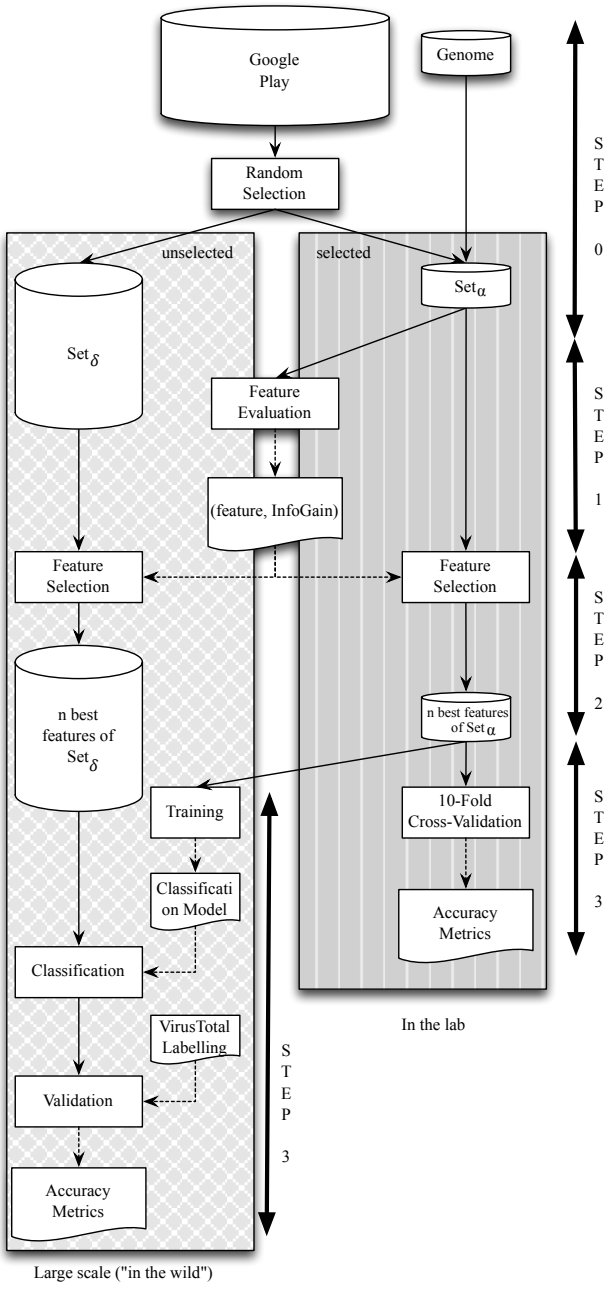

Fig. 1 The steps in our approach

Step 3: Classification validation scenarios We propose to use two distinct scenarios to validate our malware detection approach.

Validation in the lab Traditionally, machine learning-based approaches are assessed in a cross validation scenario that validates the classification model by assessing how the result will generalize to an independent dataset. To estimate how the prediction model will perform in practice, a cross-validation scenario partitions the sample data into 2 subsets. The first subset is used for learning analysis, i.e., building the model during the training phase. The second subset is used to validate the model. However, to reduce variability of the results, multiple rounds are performed and the results are averaged over the rounds. A well-known type of cross-validation is the 10-Fold cross validation (McLachlan et al 2005) which randomly partitions the sample data into 10 subsamples, 9 of which are used for training and 1 for validation. The process is then repeated with each subsample being used exactly once for validation. This method enables to consider all elements in the original sample for training but to have each element validated only and exactly once. For assessing our malware detection approach with the 10-Fold cross validation scheme we consider Set_α, which was defined in *Step 0*, as the dataset where both training and testing data will be drawn. This dataset contains both malware and goodware. Every Android application of this dataset will then be classified exactly once, allowing us to easily determine the performance of our approach in this setting.

Another common aspect of *in the lab* validation is the size of the dataset, usually a few thousands applications at most as can be seen in table 1 in appendix.

Validation in the wild. Unfortunately, the 10-Fold cross validation scenario as it is described above does not quite capture the real-world settings in which the malware detector is intended to be used. Indeed, by splitting a dataset in 10 parts, 9 of which are used for training, a 10-Fold cross-validation implicitly assumes that 90% of the domain knowledge is known beforehand—a condition that contradicts the very idea of *in the wild*.

A 10-Fold cross-validation experiment only serves to validate that a given classifier performs well in this one set of conditions, and not that its performance can be generalised outside the scope of these datasets. In the wild, the malware detection tool will only know a size-constrained sample of malware. It could also know a few true goodware, the majority of applications being of an unknown class. To detect malware in this last category, the malware detection tool must be able to perform at large.

We perform large-scale experiments where the classification algorithm of our approach is trained on Set_α. To investigate the impact of the quality of the training set, we perform two rounds of experiments where the randomly selected "goodware" from the Google Play dataset are alternatively just considered as such, or confirmed and cleaned, as true goodware using antivirus products. The trained classifier obtained is then used to predict the class, either *malware* or *goodware*, of every single application from Set_δ. Those predictions are finally compared to our reference malware classification obtained from VirusTotal to assess the performance of the approach in the wild.

4.3 Varying & Tuning the Experiments

In this section we succinctly describe the parameters that are used in our experiments to vary and tune the experiments to share insights in the practice of malware detection with machine learning techniques. These parameters were selected in accordance with the research questions outlined previously in Section 3.2.

Goodware/Malware ratio We see a first parameter in the building of the datasets. Indeed, given that the size of the malware set is fixed and known, what size of goodware should be selected in the very large set of goodware available to yield a *good ratio*? We performed various experiments to analyze the impact of the potential class imbalance between in the dataset, tuning the ratio value to 1/2, 1, 2 and up to 3, representing respectively 620, 1 247, 2 500 and 3 500 Android applications selected in the goodware set. Having the vast majority of examples from one of the classes, aka class imbalance, is a well-documented threat to Machine Learning performance in general (Van Hulse et al 2007; He and Garcia 2009). This threat is even more severe in malware detection because of the relative scarcity of malware in comparison to the number of available benign applications. Yet, surprisingly, the literature of machine learning-based malware detection often eludes this question in experiments (Cf. Section 7).

Volume of processed features Feature selection is an important step of the classification model. However, it can bias the output of the classification depending on the threshold that is set for defining *best* features. We investigate the role played by the number of features considered as relevant for our malware detector. To this end, we vary this number for the values of 50, 250, 500, 1 000, 1 500, 5 000.

Classification algorithm Last, as introduced in the description of the classification model, our malware detectors are implemented using 4 different algorithms which are well-known in the community of machine learning. For all algorithms, we have used existing implementations in Weka, namely RandomForest, J48, JRip and LibSVM, that were already referred to in the literature. In all of our experiments, these algorithms are used with the default parameters set by the Weka framework.

Overall, since the selection of Goodware performed in Step 1 of the classification is performed randomly, we reduce variability of the results by repeating 10 times each experiment with a given triplet of parameter values. In total, 4 (values for number of Goodware) $\times$6 (values for number of features) $\times$4 (number of algorithms) $\times$10 = 960 runs were processed for our experiments. The entire process took over thirty (30) CPU-days to complete.

5 Assessment

In this section we present an extensive assessment of our machine learning-based malware detection approach. We first validate the approach using a typical *in the lab* validation scenario, while discussing the impact of the different parameters that are involved in the process. Second, we compare the performance of our malware detector with approaches in the literature to highlight the relevance of our feature set. However, we take the experiments further to investigate the capability of malware detectors to scale in the wild.

5.1 Evaluation *in the lab*

We run 960 10-Fold cross validation experiments with all combinations of parameter values to assess the performance of our malware detection approach. Because in each

experiment the goodware set is varied, computed features vary, and thus the classification model leads to distinct classifiers. The validation thus assesses altogether the 960 classifiers that were built in the experiments. Figure 2 depicts the distribution of *precision*, *recall* and *F-measure* that the validation tests have yielded. In each boxplot diagram presented, whiskers go from the minimum value recorded to the maximum value. The box itself is built as follows: the bottom line of the box represents the 25^{th} percentile; the top of the box represents the 75^{th} percentile; the line inside the box represents the median value.

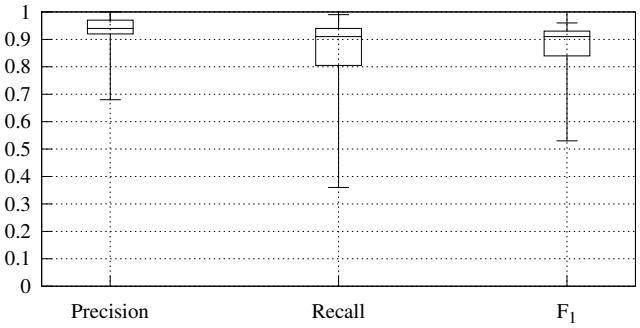

Fig. 2 Distribution of precision, recall and F-measure for the *malware* class yielded in all 960 *in the lab* experiments

Overall, the results indicate that the vast majority of our 960 built classifiers exhibit a very high precision rate with a median value of 0.94. The median value of recall is recorded at 0.91, meaning that half of the classifiers have recall values that are equal or higher to 0.91. Although recall values are lower than precision values, a large portion of the built classifiers exhibit a high recall rate. Given the precision and pecall rates, the F-measure values obtained are globally high, going from 0.53 to 0.96, with a median value of 0.91.

5.1.1 Impact of class imbalance

We now investigate in detail how class imbalance in the constructed dataset threatens the performance of machine learning-based malware detectors, and thus, how a collection of unrealistic datasets can bias validation results. To this end, as announced in Section 4.3, we perform *in the lab* experiments using datasets where the goodware/malware ratio is varied between 1/2 and 3. All other parameters are varied across all their value ranges.

Figure 3 shows that when the goodware/malware ratio is increasing in favor of goodware, the precision of malware detectors increases, while its recall decreases. The increase of the precision can be attributed to the fact that the classification model has a better view of the universe and can discriminate more accurately malware against goodware. However, at the same time, the classifiers can no longer recognize all malware since most will be more similar to some of the too many goodware. This drop in recall rate is so marked that the overall performance, measured with F-measure, decreases as revealed by the boxplots of Figure 3. This observation is of particular importance

in the field of malware detection since, in real-world scenarios, there is much more goodware than malware.

RQ1: *The performance of the machine learning-based malware detector decreases when there are fewer malware than goodware in the training dataset.*

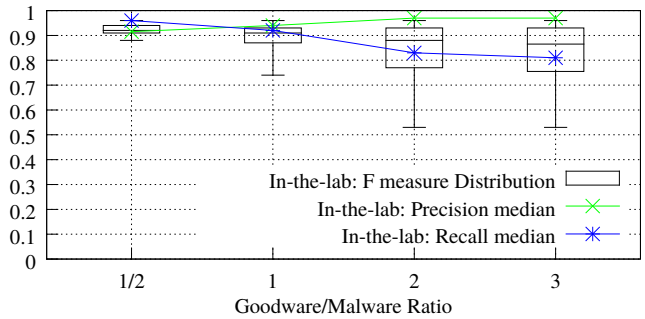

Fig. 3 Distribution of F-measure and evolution of precision and recall for various goodware/malware ratio values

5.1.2 Sensitivity to the volume of relevant features

We survey the effect that an implementation choice on the number of relevant features to retain for classification can have on the performance of the malware detector. In each experiment, about 2.5 millions distinct features are generated, most of which are evaluated to being completely irrelevant. Using the remaining features, we successively select between 50 and 5 000 to use as relevant features for the classifiers. Figure 4 shows that the overall performance, measured with F-measure, is improving with the number of features retained. However the figure also shows that over a certain threshold number, about 1 000, of features, the median value of F-measure is no longer affected. The improvement is thus confined at the upper level.

RQ2: *The more features are considered for the training phase, the better the performance of the malware detector.*

5.1.3 Effect of classification algorithm

Finally, we investigate the role played by the classification algorithm in the variation of performance between classifiers. To that end we compare the performance of classifiers after regrouping them by the underlying algorithm. Figure 5 represents the distribution of F-measure for the 4 algorithms that are used in our experiments. RandomForest, the RIPPER rule-learning algorithm, and *C4.5* exhibit high F-measure rates. SVM on the other hand provides results with a wider distribution and an overall lower F-measure.

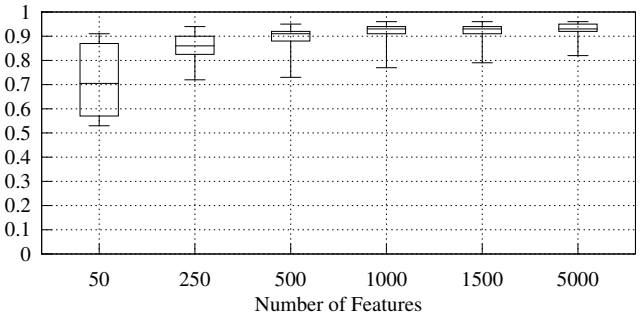

Fig. 4 Distribution of F-measure for different volumes of the set of considered relevant features

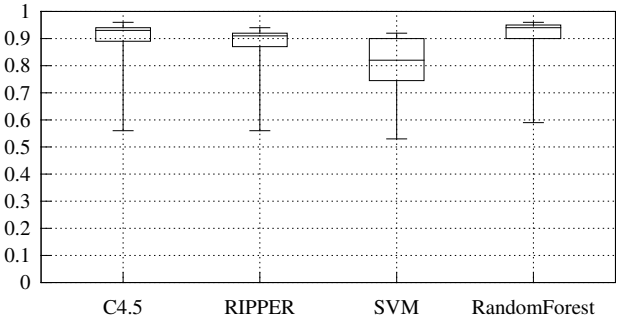

Fig. 5 Distribution of F-measure for 4 different classification algorithms

Figure 6 plots the values of precision and recall for all classifiers built when using each algorithm. We note that SVM leads to numerous classifiers with precision values close to 1, but that present lower recall rates than the other algorithms. Although SVM yields the best classifiers—the top 66 classifiers with highest precision and the top 42 with highest recall are based on SVM—it tends in our approach to yield few classifiers that have both good precision and good recall.

> **RQ3:** *Four common classifications algorithms have led to similar performance with our feature set, suggesting that the approach is not tailored to a specific algorithm.*

5.2 Comparison with Previous work

Table 1 in appendix summarizes a number of state-of-the-art machine learning-based malware detection approaches for the Android platform. We indicate the features that are used, the type of validation that were performed in the paper, the sizes and composition of the training set, the size of the testing set, if known, and an overall performance comparison with our approach. Overall, we note that our cross validation experiments have yielded at worst similar performance than state-the-art approaches, and at best, our worst classifiers perform better than classifiers of approaches in the literature. All

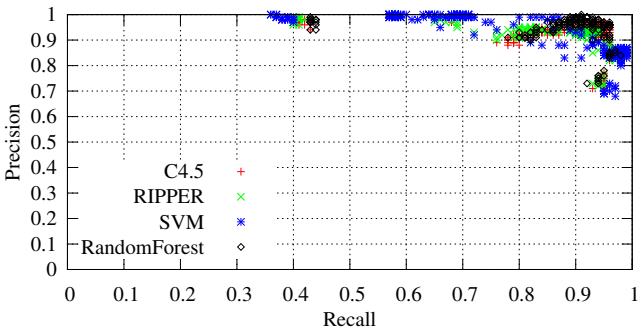

Fig. 6 Precision and recall values yielded by all classifiers for the 4 different classification algorithms

comparisons were done on equivalent experiments, i.e., with similar training and testing sets, and the same classification algorithms whenever possible.

We provide this comparison to provide a settings for a stronger, and more general, discussion on the scope of 10-Fold cross validation for approaches that are meant to be applied on datasets in the wild.

> **Finding:** *Our classifiers, when built with similar parameters than existing approaches, and evaluated* in the lab*, are highly performant.*

5.3 Evaluation in the wild

Beyond simply demonstrating the performance of our malware detection approach using cross-validation, we explore in this section its performance in the wild. We perform large-scale experiments on sizes of datasets that are unusually large for the literature of malware detection, but that better reflect realistic use-cases. Two points should be highlighted:

- 10-Fold cross-validation assesses the performance of a classifier by considering 90% of the dataset for training, thus supposing a prior knowledge of the malware class of each application in 90% of the dataset. Real-world datasets of applications however present a contrasting specificity: the known malware set is limited and is insignificant compared to the rest (i.e. goodware + unknown malware).
- Performance assessment of malware detectors should be carefully performed so as to expose the scope in which they can be of use in real-world settings. Thus, large-scale experiments with varying parameters can help refine a methodology for using, in realistic settings, a malware detection approach that was shown successful with 10-Fold cross validation on controlled datasets.

The experimental protocol used in this evaluation is similar to that used in the validation experiments of Section 5.1, except that we do not perform 10-Fold cross validation. Instead, we use our entire Training data, i.e., the entire set of known malware + a randomly selected subset of the goodware, to build the classification model (cf.

Figure 1). By varying the different parameters explicited in Section 4.3, we obtain again 960 classifiers that will be used to test the large remaining set of goodware containing from 48 422 to 51 302 applications. Each experiment with a specific set of parameters is repeated 10 times to stabilize the results. Indeed, since step 0 of our experimental setup randomly selects parts of the training dataset, repeating experiments ten times, each with a different training-set prevents the results from being biased by the possibility that the randomly selected training set is particularly *good* or particularly *bad*.

The predictions of the malware detector are then checked against the independent reference classification (cf. Section 3.2).

Figure 7 illustrates the distribution of precision, recall and F-measure values for the 960 classifiers that were built during the large-scale experiments. Overall, the classifiers exhibit a very low precision rate with a median value of 0.11. We have enumerated 13 classifiers with the highest precision value of 1. However, these only classified between 5 and 7 applications, thus yielding an exceedingly low recall rate. Also, most of the 960 classifiers have a recall value close to 0. Even the unique classifier which provided a 0.45 recall value had to classify half of the dataset as malware. Finally, with a low precision and an even lower recall, the global performance of the classifiers severely drops in large-scale experiments, with a majority of classifiers yielding a F-measure value close to 0.

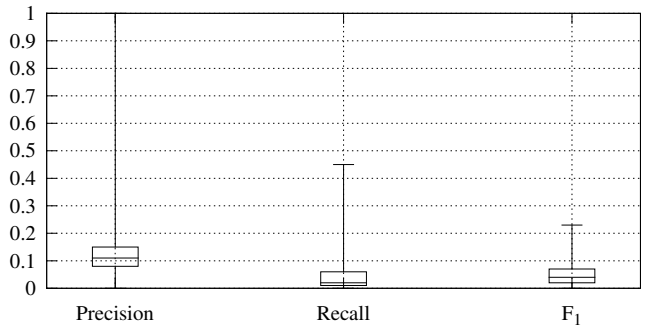

Fig. 7 Distribution of precision, recall and F-measure values in "in the wild" experiments

Figure 8 shows that when the ratio of goodware/malware in the training set is balanced in favor of the goodware set in training data, the precision rates increase slightly while recall values decrease rapidly. This figure shows that a class imbalance in favor of the goodware set leads to an overall performance drop, with the F-measure values closer to 0.

Again, as in the case of *in the lab* experiments, we investigate the sensitivity of the malware detector to the volume of relevant features. Figure 9, which depicts the distribution of F-measure values for different experiments with varied number of features that are kept as relevant, shows that, in the wild, their impact is not significant. Indeed, aside from the first boxplot for a really small number, 50, of features, all other boxplot show a compact distribution with similarly low median values.

Finally, Figure 10 presents the distribution of F-measure for classifiers built based on the four different classification algorithms used in our experiments. The distributions reveal that no algorithm significantly outperforms the others for our experiments in the wild.

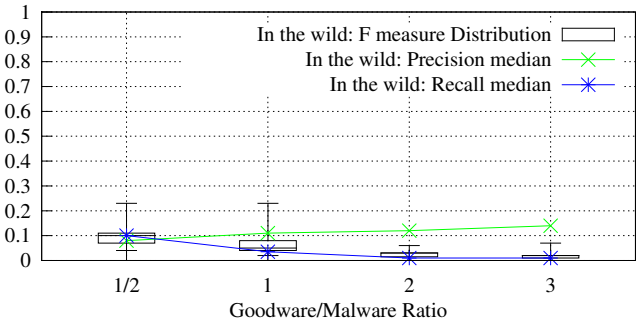

Fig. 8 Distribution of F-measure and evolution of precision and recall for various goodware/malware ratio values in "in the wild" experiments

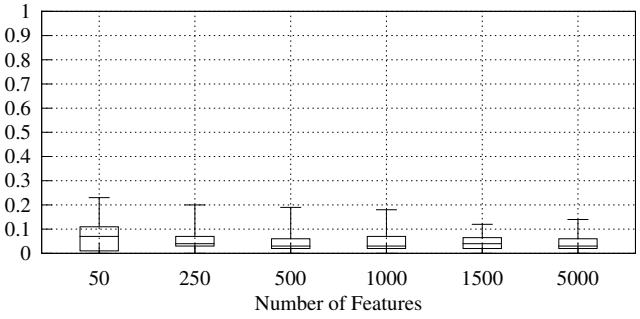

Fig. 9 Distribution of F-measure for different volumes of the set of considered relevant features in "in the wild" experiments

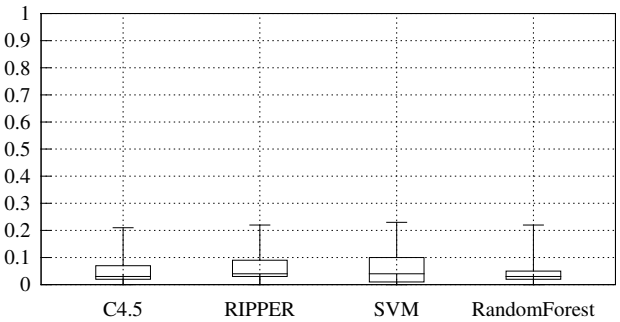

Fig. 10 Distribution of F-measure for different algorithms in "in the wild" experiments

Summary: *In the wild, experiments have revealed a poor overall performance of the malware detectors. Variations of goodware/malware ratio and classification algorithms yield the same evolutions as for* in the lab *experiments. In contrast, increase in the volume of features lead to a drop in performance during large-scale experiments.*

5.4 Discussion

In the lab experiments with the 960 different built classifiers have demonstrated that our malware detection approach performs well in comparison with existing approaches in the literature. However applying those classifiers to detect malware in very large datasets have yielded very low performance. Figure 11 illustrates the contrasting F-measure median values for both experimental scenarios with varying number of features.

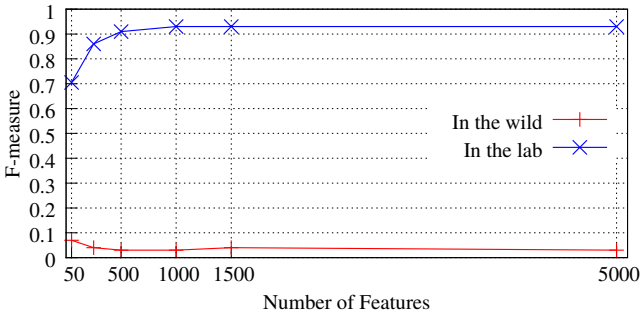

Fig. 11 Comparison of F-measure median values

We now enumerate a few points that are relevant to discuss the performance of Malware classifiers in the wild:

Size of training sets: Given the importance of the training phase, it could be argued that the size of training set that we have used in large-scale experiments are too small compared to the size of the testing set. Nonetheless, the gap between these sizes is in respect with real-world scenarios as discussed in Section 2. Furthermore, our experiments, illustrated in Figure 8, have shown that the Recall rates actually decreases when the size of training set increases.

Quality of training sets: The poor performance of classifiers during experiments in the wild could be attributed to some potential noise in the "goodware" set collected from Google Play; i.e., some goodware in this set are actually unknown malware whose features are biasing the classification model. Indeed, according to detection reports from VirusTotal, 16% of the applications obtained from Google Play are malware. We have then run experiments where the training data contained alternatively a goodware set that were uncleaned and a goodware set that were cleaned with Antivirus products. Figure 12 shows the slight improvement that cleaned dataset provides. Nonetheless, the global performance remains significantly low. Furthermore, since, to the best of our knowledge, there is no publicly available collection of known goodware that one can rely upon, a good classifier should perform relatively well even in presence of noisy training datasets.

> **RQ4:** *The machine learning-based malware detector is sensitive to the quality of training data. A cleaned goodware set positively impacts overall performance.*

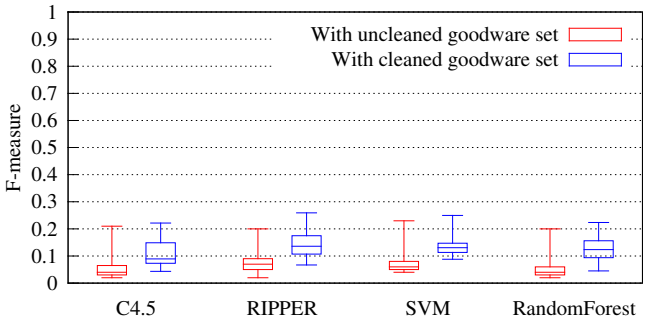

Fig. 12 Distribution of F-measure values with cleaned and uncleaned goodware sets for experiments in the wild

6 Threats to Validity

Our study presents a number of threats to validity that we discuss in the following to highlight their potential impact on our findings and the measures we have taken to mitigate their bias.

6.1 External Validity

Datasets representativity: During collection of datasets from Google Play, we did not consider downloading any paid application. However, free applications account for the majority of Android applications available (AppBrain 2013a) and appear to be the most affected by malware.

Furthermore, the malware from the Genome dataset that we have used may not be representative enough of the malware corpus available in Google Play. However, to the best of our knowledge, this is the most comprehensive collection of Android malware available to researchers in the Security and Privacy field. Besides, malware representativity is hard to define in practice, since it would require that one knows beforehand all malware that are being looked for.

Google's own malware detector: In February 2012, Google announced (Google 2012) they were using *Bouncer*, their own Android malware detector, to prevent malicious applications to reach the official Google Play market. While Bouncer still allows many malware to enter Google Play (Allix et al 2014b), it may bias our dataset collection.

Since both our *in the lab* and *in the wild* experiments used apps collected from Google Play, both validation scenarios should be affected by this bias. Bouncer therefore cannot play a significant role in the performance gap we observed. However, if Bouncer had a negative impact on Android malware detectors, our results show that this impact would be marginal *in the lab*, but significant *in the wild*, hence highlighting the importance of *in the wild* experiments.

6.2 Construct Validity

Labeling methods: In our experiments, two different reference classification sources were used as ground truth: *in the lab* experiments were based on the Genome project classi-

fication alone while *in the wild* experiments used the Genome project for training and were tested against VirusTotal classification. Although we verified beforehand that **every app from the Genome project is classified as malware by VirusTotal**, the use of two different labeling sources could be one possible explanation for the differences in accuracy we found when comparing *in the lab* with *in the wild* experiments. To investigate this hypothesis, we performed the same experiments again, this time using only VirusTotal for both training and testing. As can be seen on Fig 13, using a single, coherent reference classification does not result in significantly different results. Hence, the performance gap between *in the lab* and *in the wild* experiments cannot be explained by our usage of labelling sources.

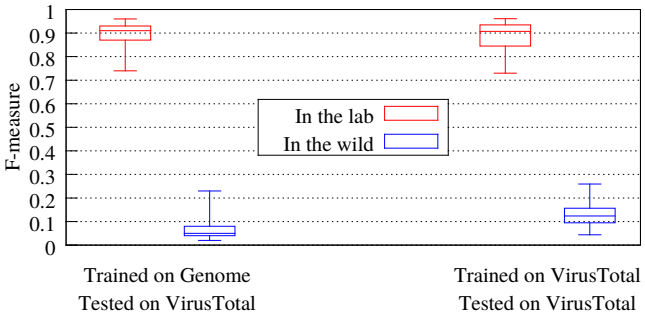

Fig. 13 Distribution of F-measure for different classification references usages

Exhaustiveness of classification algorithms: Machine-learning algorithms perform differently depending on the context. It is thus possible that the four well-known algorithms that we have selected were used in this study outside of their comfort zone. Nonetheless, we note that 3 very distinct algorithms exhibited similar patterns, suggesting that our findings are not specific to a particular type of classification algorithm.

Relevance of feature set: Our experiments were performed with the same type of features, which are based on basic blocks of CFGs. Possibly, this particular feature set is incompatible with experiments in the wild. However, we have not found in the state-of-the-art literature evidence suggesting that other feature sets with high performance in *in the lab* validation actually perform well in large-scale experiments as well.

Limited experiments with 2-grams extracted from raw bytecode, resulted in the same performance gap between *in the lab* and *in the wild* validation scenarios.

Furthermore, we note that if our feature-set was deemed unsound, or unsuitable for this study, this would actually strengthen our argument. Indeed, it would demonstrate that even an unsound feature-set can lead to high-performance *in the lab*, or in other words, that high performance *in the lab* is not even a valid indicator of soundness for a feature-set.

6.3 Internal Validity

Composition of training and testing sets: The size of training sets and the ratio between goodware and malware sets take various values that appear to be unjustified since,

to the best of our knowledge, no survey has determined their appropriate values for malware detection. However, we have ensured that the sizes that are used in our study are comparable to other research work, and that they are representative of the data available to the research community.

6.4 Other Threats

Specificity of Findings to the Android platform: Experiments in this study focused on Android applications. We have not studied malware detection for other Operating Systems. Although our feature set does not take into account any specificities, such as Android Permissions scheme, we cannot rule out that the gap between *in the lab* and *in the wild* scenarios could be narrower in other platforms.

7 Related Work

Previously, we have reported (Allix et al 2014a) preliminary findings of this work to the Computer security community.

A significant amount of Machine Learning approaches to malware detection has been presented to the research community. Although most of those approaches could not be reproduced due to undisclosed parameters and/or undisclosed datasets, we try to compare their evaluation metrics with our most-closely *in the lab* classifiers. None of the approaches introduced by the literature discussed in this section provide a large scale evaluation of their approach.

Android malware detection In 2012, Sahs and Khan (2012) 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 [6] 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 (2012) 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 (2013) leveraged dynamic application profiling in their malware detector. The evaluation metrics of their 10-Fold experiment are slightly lowers than ours. Demme et al (2013) also used dynamic application analysis to perform malware detection with a dataset of 210 goodware and 503 malware. Many of our *in the lab* classifiers achieved higher performance than their best classifier. Yerima et al (2013) built malware classifiers based on API calls, external program execution and permissions. Their dataset consists in 1 000 goodware and 1 000 malware. Many of our *in the lab* classifiers achieved higher performance than their best classifier. Canfora et al (2013) 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.

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

Windows malware detection Kolter and Maloof (2006) performed malware classification on Windows Executable files. Using n-grams extracted from those 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 malwares and 561 goodware, the second one containing 1 651 malware and 1 971 goodware. Many of our *in the lab* classifiers achieved higher performance metrics. In 2006, Henchiri and Japkowicz (2006) provided experimental results of a malware detector based on a sophisticated n-grams selection algorithm. They evaluated their classifier using 5-Fold[7] 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 (2007) 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 et al (2001) 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 (2008a) 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 (Perdisci et al 2008b) 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 (2012) recently presented "Mal-ID", 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.

8 Conclusion

We have discussed in this paper the validation of machine-learning malware detection with *in the lab* and *in the wild* scenarios. A first contribution of our work is a Feature set for building classifiers that yield high performance measures in *in the lab* evaluation scenarios and in comparison with state-of-the-art approaches. Beyond this evaluation, however, we have assessed the actual ability of our classifiers to detect Malware in a significantly large dataset. The recorded poor performance has provided us with new insights as to the limits to which an *in the lab* validation scheme is a reliable indicator for real-world malware detectors. We have thus identified several parameters that are likely to impact the performance of Malware Detectors. Finally, we make available to the research community all our datasets to improve the research on Android malware detection.

[7] 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.

Our Argument. By presenting here an approach that exhibits high performance *in the lab* and yet has little practical usefulness in the wild, we demonstrated that there exists at least one approach for which this performance gap exists. While this paper cannot demonstrate that the same gap exists for other published approaches, we claim that until those approaches are tested in the wild, they cannot be supposed to represent a significant improvement to the malware detection domain.

We also showed here that this issue of validation scenario is not merely a minor bias in experimental results: *in the lab* results are not a slightly optimistic version of results *in the wild*. Instead, they can be vastly different and tell widely different stories.

Hence, evaluating malware detector *in the wild*, with a sound empirical methodology is of the utmost importance. In other words, we call for the Machine Learning-based malware detection community to devise and agree on what would be sound, in-depth and meaningful validation scenarios.

In future work, we plan to investigate the reasons of the observed performance gap, and to formalise a methodology for sound, extensive, reliable and reproducible empirical evaluation of malware detectors.

Acknowledgements We would like to thank VirusTotal for providing us the ability to leverage their infrastructure and detection report databases to build a reference classification as described in section 3.2.

9 Appendix

Table 1 Recent research in Machine Learning-based Android Malware Detection

Authors	Features	Algorithm	Evaluation	Datasets	Training set	Test Set	Comment
Sahs and Khan (2012)	Permissions, CFG subgraphs	1-class SVM	k-fold	2 081 goodware 91 malware	Subsets of the goodware set	91 malware (and remainder of training set?)	Sahs & Khan approach yielded high recall with low precision. The vast majority of our *in the lab* classifiers yielded both a high recall and a high precision.
Amos et al (2013)	Profiling (Dynamic)	RandomForest, C4.5, etc.	10-fold on training set and evaluation on a test set	1 777 Apps	408 goodware 1 330 malware	24 goodware 23 malware	Our closest experiment (goodware/malware ratio: 1/2) yielded dozens of classifiers with equivalent or better performance
Yerima et al (2013)	API calls, external tool execution, permissions (Static)	Bayesian	5-fold	1 000 goodware 1 000 malware	?[1]	?[1]	Our closest *in the lab* experiment (goodware/malware ratio: 1) yielded 74 classifiers with both higher recall and higher precision than Yerima et al.'s best classifier.
Demme et al (2013)	Performance Counters (Dynamic)	KNN, RandomForest, etc.	?[1]	210 goodware 503 malware	?[1]	?[1]	The majority of our *in the lab* classifiers yielded higher recall and higher precision than Demme et al.'s best classifier
Canfora et al (2013)	SysCalls, Permissions	C4.5, RandomForest, etc.	?[1]	200 goodware 200 malware	?[1]	?[1]	In our closest experiment by dataset size (goodware/malware ratio :1/2), our worst classifier performs better than Canfora et al.'s best classifier. In our closest experiment by goodware/malware ratio (1), the vast majority of our classifier perform better than Canfora et al.'s best classifier.
Wu et al (2012)	Permissions, API Calls, etc.	KNN, NaiveBayes	?[1]	1 500 goodware 238 malware	?[1]	?[1]	More than 100 of our *in the lab* classifiers yielded both a higher recall and a higher precision than their best classifier.

References

Allix K, Bissyandé TF, Jérome Q, Klein J, State R, Le Traon Y (2014a) Large-scale machine learning-based malware detection: Confronting the "10-fold cross validation" scheme with reality. In: Proceedings of the 4th ACM Conference on Data and Application Security and Privacy, ACM, New York, NY, USA, CODASPY '14, pp 163–166, DOI 10.1145/2557547.2557587, URL http://doi.acm.org.proxy.bnl.lu/10.1145/2557547.2557587

Allix K, Jérome Q, Bissyandé TF, Klein J, State R, Le Traon Y (2014b) A forensic analysis of android malware: How is malware written and how it could be detected? In: Computer Software and Applications Conference (COMPSAC)

Amos B, Turner H, White J (2013) Applying machine learning classifiers to dynamic android malware detection at scale. In: Wireless Communications and Mobile Computing Conference (IWCMC), 2013 9th International, pp 1666–1671, DOI 10.1109/IWCMC.2013.6583806

AndroGuard (2013) Apktool for reverse engineering android applications. URL https://code.google.com/p/androguard/, accessed: 2013-09-09

AppBrain (2013a) Comparison of free and paid android apps. URL http://www.appbrain.com/stats/free-and-paid-android-applications, accessed: 2013-09-09

AppBrain (2013b) Number of available android applications. URL http://www.appbrain.com/stats/number-of-android-apps, accessed: 2013-09-09

Breiman L (2001) Random forests. Machine learning 45(1):5–32

Canfora G, Mercaldo F, Visaggio CA (2013) A classifier of malicious android applications. In: Availability, Reliability and Security (ARES), 2013 eight International Conference on

[1] We were unable to infer this information.

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

Cohen WW (1995) Fast effective rule induction. In: Machine Learning-International Workshop Then Conference, Morgan Kaufmann Publishers, Inc., pp 115–123

Cortes C, Vapnik V (1995) Support-vector networks. Machine Learning 20(3):273–297, DOI 10.1007/BF00994018, URL http://dx.doi.org/10.1007/BF00994018

Demme J, Maycock M, Schmitz J, Tang A, Waksman A, Sethumadhavan S, Stolfo S (2013) On the feasibility of online malware detection with performance counters. In: Proceedings of the 40th Annual International Symposium on Computer Architecture, ACM, New York, NY, USA, ISCA '13, pp 559–570, DOI 10.1145/2485922.2485970

Enck W, Octeau D, McDaniel P, Chaudhuri S (2011) A study of android application security. In: Proceedings of the 20th USENIX conference on Security, USENIX Association, Berkeley, CA, USA, SEC'11, pp 21–21, URL http://dl.acm.org/citation.cfm?id=2028067.2028088

Felt AP, Finifter M, Chin E, Hanna S, Wagner D (2011) A survey of mobile malware in the wild. In: Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, ACM, New York, NY, USA, SPSM '11, pp 3–14, DOI 10.1145/2046614.2046618, URL http://doi.acm.org/10.1145/2046614.2046618

Google (2012) Android and security (bouncer announcement). http://googlemobile.blogspot.fr/2012/02/android-and-security.html, accessed: 2014-06-14

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH (2009) The weka data mining software: an update. SIGKDD Explor Newsl 11(1):10–18, DOI 10.1145/1656274.1656278

He H, Garcia E (2009) Learning from imbalanced data. Knowledge and Data Engineering, IEEE Transactions on 21(9):1263–1284, DOI 10.1109/TKDE.2008.239

Henchiri O, Japkowicz N (2006) A feature selection and evaluation scheme for computer virus detection. In: Proceedings of the Sixth International Conference on Data Mining, IEEE Computer Society, Washington, DC, USA, ICDM '06, pp 891–895, DOI 10.1109/ICDM.2006.4

Jacob A, Gokhale M (2007) Language classification using n-grams accelerated by fpga-based bloom filters. In: Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications: held in conjunction with SC07, Reno, Nevada, USA, HPRCTA '07, pp 31–37

Kephart JO (1994) A biologically inspired immune system for computers. In: In Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, MIT Press, pp 130–139

Kolter JZ, Maloof MA (2006) Learning to detect and classify malicious executables in the wild. J Mach Learn Res 7:2721–2744, URL http://dl.acm.org/citation.cfm?id=1248547.1248646

McLachlan G, Do KA, Ambroise C (2005) Analyzing microarray gene expression data, vol 422. Wiley. com

Perdisci R, Lanzi A, Lee W (2008a) Classification of packed executables for accurate computer virus detection. Pattern Recognition Letters 29(14):1941 – 1946, DOI 10.1016/j.patrec.2008.06.016, URL http://www.sciencedirect.com/science/article/pii/S0167865508002110

Perdisci R, Lanzi A, Lee W (2008b) Mcboost: Boosting scalability in malware collection and analysis using statistical classification of executables. In: Computer Security Applications Conference, 2008. ACSAC 2008. Annual, pp 301–310, DOI 10.1109/ACSAC.2008.22

Pieterse H, Olivier M (2012) Android botnets on the rise: Trends and characteristics. In: Information Security for South Africa (ISSA), 2012, pp 1–5, DOI 10.1109/ISSA.2012.6320432

Pouik, G0rfi3ld (2012) Similarities for fun & profit. Phrack 14(68), URL http://www.phrack.org/issues.html?id=15&issue=68

Quinlan JR (1993) C4. 5: programs for machine learning, vol 1. Morgan kaufmann

Rossow C, Dietrich C, Grier C, Kreibich C, Paxson V, Pohlmann N, Bos H, van Steen M (2012) Prudent practices for designing malware experiments: Status quo and outlook. In: Security and Privacy (SP), 2012 IEEE Symposium on, pp 65–79, DOI 10.1109/SP.2012.14

Sahs J, Khan L (2012) A machine learning approach to android malware detection. In: Intelligence and Security Informatics Conference (EISIC), 2012 European, IEEE, pp 141–147,

DOI 10.1109/EISIC.2012.34
Santos I, Penya YK, Devesa J, Bringas PG (2009) N-grams-based file signatures for malware detection. In: ICEIS, pp 317–320
Schultz M, Eskin E, Zadok E, Stolfo S (2001) Data mining methods for detection of new malicious executables. In: Security and Privacy, 2001. S P 2001. Proceedings. 2001 IEEE Symposium on, pp 38–49, DOI 10.1109/SECPRI.2001.924286
Tahan G, Rokach L, Shahar Y (2012) Mal-id: Automatic malware detection using common segment analysis and meta-features. J Mach Learn Res 98888:949–979
Van Hulse J, Khoshgoftaar TM, Napolitano A (2007) Experimental perspectives on learning from imbalanced data. In: Proceedings of the 24th international conference on Machine learning, ACM, New York, NY, USA, ICML '07, pp 935–942, DOI 10.1145/1273496.1273614
Wu DJ, Mao CH, Wei TE, Lee HM, Wu KP (2012) Droidmat: Android malware detection through manifest and api calls tracing. In: Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on, pp 62–69, DOI 10.1109/AsiaJCIS.2012.18
Yerima S, Sezer S, McWilliams G, Muttik I (2013) A new android malware detection approach using bayesian classification. In: Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on, pp 121–128, DOI 10.1109/AINA.2013.88
Zhang B, Yin J, Hao J, Zhang D, Wang S (2007) Malicious codes detection based on ensemble learning. In: Proceedings of the 4th international conference on Autonomic and Trusted Computing, Springer-Verlag, Berlin, Heidelberg, ATC'07, pp 468–477
Zhou Y, Jiang X (2012) Dissecting android malware: Characterization and evolution. In: Proceedings of the 2012 IEEE Symposium on Security and Privacy, IEEE Computer Society, Washington, DC, USA, SP '12, pp 95–109, DOI 10.1109/SP.2012.16, URL http://dx.doi.org/10.1109/SP.2012.16