Noname manuscript No.
(will be inserted by the editor)

Assessing the opportunity of combining state-of-the-art Android malware detectors

Nadia Daoudi[1] · **Kevin Allix**[2] · **Tegawendé F. Bissyandé**[1] · **Jacques Klein**[1]

Received: date / Accepted: date

Abstract Research on Android malware detection based on Machine learning has been prolific in recent years. In this paper, we show, through a large-scale evaluation of four state-of-the-art approaches that their achieved performance fluctuates when applied to different datasets. Combining existing approaches appears as an appealing method to stabilise performance. We therefore proceed to empirically investigate the effect of such combinations on the overall detection performance. In our study, we evaluated 22 methods to combine feature sets or predictions from the state-of-the-art approaches. Our results showed that no method has significantly enhanced the detection performance reported by the state-of-the-art malware detectors. Nevertheless, the performance achieved is on par with the best individual classifiers for all settings. Overall, we conduct extensive experiments on the opportunity to combine state-of-the-art detectors. Our main conclusion is that combining state-of-the-art malware detectors leads to a stabilisation of the detection performance, and a research agenda on how they should be combined effectively is required to boost malware detection. All artefacts of our large-scale study (i.e., the dataset of ∼0.5 million apks and all extracted features) are made available for replicability.

Keywords Android · Malware · Machine Learning · Ensemble Learning

1 Introduction

Early 2021, an Antivirus provider has reported having flagged more than 1.4 million malware apps during the first quarter of 2021, which represents an

[1] SnT, University of Luxembourg 29, Avenue J.F Kennedy, L-1359, Luxembourg, Luxembourg. E-mail: firstname.lastname@uni.lu
[2] CentraleSupelec, Avenue de la Boulaie, CS 47601, F-35576 Cesson-Sévigné Cedex, France. E-mail: kevin.allix@centralesupelec.fr

increase of 298 998 malware compared to the same quarter of 2020 (Kaspersky 2021). Moreover, a recent investigation made by a security company has reported that an Android malware app is published every eight seconds, indicating that malware is growing at an alarming rate (G DATA 2020).

Due to its ability to learn automatically from input data, Machine Learning techniques have been extensively leveraged to develop approaches for Android malware detection (Arp et al. 2014; Onwuzurike et al. 2019; Garcia et al. 2018; Wu et al. 2019, 2012; Fereidooni et al. 2016; Afonso et al. 2015). In the literature, several of such approaches produced detectors that were reported to be highly effective, and each new publication claims to now achieve state-of-the-art performance. In 2014, DREBIN (Arp et al. 2014) has made a breakthrough in Android malware detection by proposing an approach that detects malware using a large variety of app features. Three years later, MaMaDroid (Mariconti et al. 2017) has been proposed and has claimed to be more generic and robust than DREBIN. Other approaches have followed the same ongoing trend by reporting detection scores that are all above 90%. The spread of Android malware, however, hints that the challenges of detection remain intact for practitioners. This situation calls for a thorough revisitation of Android malware detection literature. A first step in this direction is to conduct independent evaluations of state-of-the-art malware detectors to highlight limitations and opportunities for improvement.

When facing weak detectors (e.g., state-of-the-art approaches that are not effective under all settings), an immediate solution could be to investigate their combination, e.g., using Ensemble Learning (Brown 2010). Ensemble Learning has been evaluated in the literature of malware detection using some selected features and ML algorithms (Yerima et al. 2015; Zhang and Jin 2016; Zhao et al. 2018; Zhu et al. 2020; Zhang et al. 2015). Researchers have then relied on Ensemble Learning techniques to propose independent approaches for malware detection. These approaches are developed by selecting a set of features (e.g., permission and intents (Idrees et al. 2017)) and a method to combine the base learners (e.g., Majority Voting (Christianah et al. 2020)). To date, however, no study has considered combining existing state-of-the-art Android malware detectors in an attempt to advance the research field in a clear and principled manner. Unfortunately, this focus on proposing new detectors without first thoroughly assessing and building on existing work impedes the progress of the research domain.

This paper. Building on large-scale datasets and re-executing existing approaches, we empirically show that state-of-the-art Android malware detectors yield performance results that significantly depend on the evaluation dataset. Indeed, none of the studied approaches has been reported to reach the highest prediction performance on all the evaluation settings. This finding suggests that trusting a single approach in a real world setting is unrealistic. To overcome this limitation, we investigate whether the combination of state-of-the-art Android malware detectors can yield better detection performance. We build on a recent study on the Reproducibility/Replicability of ML-based Android malware detectors (Daoudi et al. 2021b) which has considered research

works from 16 major venues in Machine Learning, Security, and Software Engineering. In this study, Daoudi et al. (2021b) were able to successfully reproduce/replicate four state-of-the-art malware detectors. These detectors will be used as the basis for our work.

Our work assesses the impact of combining the best approaches from the literature, each of which contributing with a specific way of modelling Android apps. Specifically, DREBIN extracts eight types of string features from the Manifest file and the DEX bytecode, including permissions, intents, hardware components, and suspicious API calls. On the other hand, MaMaDroid models the behaviour of the apps using Markov Chains representation of the abstracted API calls. As for RevealDroid (Garcia et al. 2018), it focuses on features related to API usage, reflection, and native calls. Finally, MalScan (Wu et al. 2019) borrows methods from social networks analysis to detect malware. This approach models the call graph of the app as a social network graph and performs different centrality analyses. Each of the studied approaches relies on an ML algorithm that performs best with its set of features since their authors have already evaluated and configured them using the best hyper-parameters. Such efforts need to be exploited to further advance Android malware detection research. To this end, we set a research agenda to assess the value of combining, with Ensemble Learning, the features sets or the predictions proposed in state-of-the-art approaches. We rely on Ensemble Learning to mitigate the dependence of individual approaches on the evaluation settings. Our work evaluates the four state-of-the-art approaches on two large datasets of over 197k and 265k apps and studies the impact of either combining their feature sets or combining the detectors themselves using Ensemble Learning.

Overall, we make the following contributions:

- We conduct a comparative evaluation of four state-of-the-art Android malware detectors (+ variants) using the same experimental setup to identify the best performing approach. The studied detectors are: DREBIN, MaMaDroid (two variants of the approach), RevealDroid, MalScan (six variants of the approach).
- We examine the similarities/difference in the malware detected by state-of-the-art approaches.
- We investigate the impact of merging *feature sets* from state-of-the-art Android malware approaches on the detection performance.
- We investigate the impact of combining *predictions* from state-of-the-art malware detectors using 16 combination methods.

Our work has resulted in the following findings:

❶ The performance of state-of-the-art Android malware detectors is highly dependent on the experimental dataset. None of the studied approaches has reported the best detection performance on all the evaluation settings.
❷ Some families of malware are detected very accurately by some state-of-the-art approaches, but almost completely escape detection of some other approaches.

❸ Combining features and predictions from state-of-the-art malware detectors (i.e., using `Bagging` and `Ensemble Selection`) is promising to leverage the capabilities of the best detectors and maintain a stable detection rate on all the evaluation settings.

2 Study Design

In this section we introduce the research questions, present the datasets, overview the experimental setup and enumerate the state-of-the-art malware detectors that are leveraged.

2.1 Research Questions

In previous works, Allix et al. (2016a) then Pendlebury et al. (2019) have presented study results which suggest that literature evaluation of Android malware detection approaches generally suffers from spatial and temporal biases. Most of the times, each approach is assessed only on a specific dataset, with limited comparison to existing work. Thus, there is a missed opportunity to definitively understand the contribution of each approach and eventually build up on existing works for improved detection.

Given that each new approach claims to outperform others, a first step towards addressing the biased comparison issues would be to undertake an independent and fair assessment of state-of-the-art approaches in order to compare their results under different settings:

– **RQ1:** Is there a state-of-the-art malware detector that outperforms all others across all datasets?

To further investigate the similarities and differences between state-of-the-art malware detectors, a possible direction would be to examine the similarities and differences in the malware detected by these approaches.

– **RQ2:** To what extent do state-of-the-art approaches detect similar/different malware?

In the literature, authors often insist on the engineering of a new feature set, but do not generally investigate in detail the added value of their feature set compared to previous approaches. We hypothesise that if each feature set brings its own value, combining them should noticeably improve the detection performance.

– **RQ3:** Does merging the *feature sets* from state-of-the-art approaches lead to a high-performing malware detector in all the settings?

Another way of exploiting the combined value of different approaches would be to consider each approach as a whole. Instead of combining feature sets before classifier training, we can combine prediction results after training each approach (i.e., feature set + algorithm) independently.

– **RQ4:** Does combining *predictions* from state-of-the-art approaches lead to a high-performing malware detector in all the settings?

Finally, we statistically compare the detection performance of state-of-the-art approaches and the classifiers produced by the combination of features and predictions:

– **RQ5:** Does combining *feature sets* or *predictions* from state-of-the-art approaches lead to classifiers that significantly outperform the original detectors?

2.2 Dataset

Our study considers two main datasets, which are summarised in Table 1:

`Literature dataset` This dataset is constructed by collecting app samples used in the literature[1] to validate state-of-the-art malware detection approaches (cf. Section 2.4). Apps in this dataset span from 2010 to 2018. Overall, the literature dataset includes 43 819 malware and 153 616 benign apps. In our experiments, we evaluate our classifiers not only on the whole literature dataset, but also on each of its subsets separately (i.e., the dataset used for DREBIN, for MaMaDroid, for RevealDroid, and for MalScan).

`AndroZoo dataset` This dataset is collected from the AndroZoo (Allix et al. 2016b) repository whose maintainers continuously crawl Android apps from different sources (including Google Play, AppChina, etc.). We consider that an app is labelled as benign if it has not been detected by any Antivirus engine from VirusTotal[2]. Following up on previous work (Arp et al. 2014), we consider an app to be a malware if it has been detected by at least two Antivirus from VirusTotal. For this dataset we focused on recent apps created[3] in 2019 and 2020. Overall our AndroZoo dataset includes 78 002 malware samples and 187 797 benign samples. Similarly, we also conduct our experiments on the whole AndroZoo dataset as well as its subsets (i.e., 2019 and 2020 subsets).

2.3 Experimental setup

All experiments (to evaluate the literature approaches, the merged feature sets, and the combinations of predictions) are performed on each of the collected datasets. We consider two evaluation scenarios per experiment and per dataset, following up on previous work (Pendlebury et al. 2019; Allix et al. 2015) which highlighted biases in empirical evaluation of machine learning-based malware detection:

[1] Since some of these apps are not shared by their original authors, we rely on the replicated datasets described in the reproduction study (Daoudi et al. 2021b)

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

[3] We consider the compilation dates

Table 1: Dataset summary

	Subsets	**Malicious apps**	**Benign apps**	**Total**
`Literature dataset`	DREBIN	5363	111 592	116 955
	MaMaDroid	30 895	7756	38 651
	RevealDroid	18 924	22 480	41 404
	MalScan	12 943	14 038	26 981
	Total**	43 819	153 616	197 435
`AndroZoo dataset`	2019	59 256	122 966	182 222
	2020	18 746	64 831	83 577
	Total	78 002	187 797	265 799

** The total is calculated after removing redundant apps

- `Temporally-consistent`: The classifiers are trained on old apps, and tested on new apps (i.e., the dataset is split based on the apps' creation dates).
- `Temporally-inconsistent`: the classification experiment does not take into account the creation time of the apps (i.e., the dataset is shuffled before the split into training, validation, and test sets)

In our experiments, each dataset is split into training (80%), validation (10%), and test (10%). For the `Temporally-inconsistent` settings, we repeat each experiment ten times after randomly shuffling and splitting the datasets. As for the `Temporally-consistent` settings, we also repeat the experiments ten times by randomly selecting 90% of the apps from the training, validation, and test splits (i.e., the training and the test contain the oldest and the newest apps, respectively). For both settings we report an average detection performance.

We rely on Recall, Precision, F1-score, and the Accuracy to measure the classification performance. In our evaluation, we use these metrics to refer to the average detection scores since our experiments are repeated ten times. We present the formulas of these metrics below:

$$Precision = \frac{TP}{TP + FN} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$F_1 score = 2. \frac{precision \,.\, recall}{precision + recall} \tag{3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4}$$

All the algorithms trained on the merged feature sets (c.f., Section 3.3) or trained to combine the predictions (c.f., Section 3.4) are used with their default parameters provided by the scikit-learn framework[4].

[4] `https://scikit-learn.org`

2.4 Study subjects: literature detectors

Our work builds on four state-of-the-art Android malware detectors presented at major venues. These approaches have been identified in a recent study (Daoudi et al. 2021b) that assessed the reproducibility/replicability of Machine Learning-based Android malware detection approaches in the literature. We have considered these malware detectors for two main reasons:

- Indeed, a tremendous number of malware detection papers are published in the literature, but our study focuses on the best approaches with the most significant contributions in the field. Thus, our study subjects are selected among papers published in 16 top venues in Software Engineering, Security, and Machine Learning: EMSE, TIFS, TOSEM, TSE, FSE, ASE, ICSE, NDSS, S&P, Usenix Security, CCS, AsiaCCS, SIGKDD, NIPS, ICML, and IJCAI.
- In order to accurately and fairly assess the detection performance of the studied approaches, they need to be *reproducible*. Specifically, our evaluation results can be attributed to the original approaches only in the case when the reproducibility of these detectors is verified and confirmed. In the reproduction study from which we select our approaches (Daoudi et al. 2021b), ten years of Android malware detection papers from major venues have been considered. However, only four approaches have been successfully reproduced. Our study subjects are the only state-of-the-art malware detectors whose reproducibility has been validated in the literature.

We present below a brief description of these approaches. We also represent in table 2 a summary of the features and ML algorithms used by these approaches and we refer the reader to the reproduction study, or the original papers for further details.

2.4.1 DREBIN *(Arp et al. 2014)*

It trains a LinearSVC classifier using eight types of features that are extracted from the DEX and the Manifest files: hardware components, requested permissions, app components, filtered intents, restricted API calls, used permissions, suspicious API calls, and network addresses.

2.4.2 MaMaDroid *(Mariconti et al. 2017)*

For each app, it first generates a call graph with abstracted API calls to then build a feature vector. MaMaDroid proposes two variants to abstract the API calls: either by only considering their package name (MaMaDroid Package model), or by considering the first component of their package name (MaMaDroid Family model). In both cases, the Markov Chain representation of the abstracted API calls is then used to create the feature vectors. The two variants of the approach (i.e., MaMaDroid Family and MaMaDroid Package) rely on Random Forest (RF) classifier.

2.4.3 RevealDroid *(Garcia et al. 2018)*

It trains a LinearSVC classifier using three types of features: Android API usage (Number of invocations of Android API methods and packages), Reflective, and Native Call features.

2.4.4 MalScan *(Wu et al. 2019)*

It represents the call graph of the apps as a social network to perform centrality analysis. Six variants of this approach are proposed and they are all trained using KNN algorithm. The type of the model is determined by the type of the centrality measure: MalScan Degree, MalScan Katz, MalScan Closeness, MalScan Harmonic, MalScan Average (it uses as features the average of the feature vectors from the four previous models) and MalScan Concatenate (the feature vectors are the concatenation of the feature vectors from MalScan Degree, MalScan Katz, MalScan Closeness, MalScan Harmonic).

Table 2: Study subjects

	Features set	**ML algorithm**
DREBIN	hardware components, requested permissions, app components, filtered intents, restricted API calls, used permissions, suspicious API calls, and network addresses	LinearSVC
MaMaDroid	Markov Chain representation of the abstracted API calls	Random Forest
RevealDroid	Android API usage, Reflective, and Native Call Features	LinearSVC
MalScan	Centrality analysis on the social network representation of the call graph	KNN

3 Study Results

To answer our research questions, we have conducted our experiments using `Literature dataset`, `AndroZoo dataset`, and their subsets.

3.1 **RQ1:** Is there a state-of-the-art malware detector that outperforms all others across all datasets?

Android malware detectors in the literature are usually evaluated using different experimental setups and datasets. In this section, we aim to assess the performance of the state-of-the-art malware detection approaches under

consistent experimental conditions. Specifically, we evaluate the effectiveness of the four state-of-the-art malware detectors (and their variants) using the datasets described in Section 2.2 and under the experimental setup described in Section 2.3. For conciseness, we use, for instance, LitTempInconsist to refer to the experimental setup where we use a `Literature dataset` in a `temporally-inconsistent` experiment. Table 3 describes our experimental settings.

Table 3: Summary of our experimental setting

	`Temporally-inconsistent`	`Temporally-consistent`
`Literature dataset`	LitTempInconsist	LitTempConsist
`AndroZoo dataset`	AndTempInconsist	AndTempConsist

Since we consider five `Literature datasets` and three `AndroZoo datasets` (i.e., whole datasets and their subsets), the total number of our experimental settings reaches 16. In the remainder of this paper, we use "dataset" and "setting" interchangeably.

We report the average F1 score for the considered malware detection approaches in the upper part of Table 4. We also present the Recall, Precision, and Accuracy scores of our experiments in Table 7, Table 8, Table 9, and Table 10 in the Appendix.

We observe that the performance of the classifiers varies considerably across datasets. On the whole LitTempConsist, all the approaches have reported detection scores that are significantly low, with a best F1 score of 0.44. This result is consistent with the finding of Tesseract (Pendlebury et al. 2019) on a `temporally-consistent` setting. For the other datasets (i.e., `AndroZoo datasets` and `Literature subsets`), the detection performance on the `temporally-consistent` experiment is also generally lower than the performance reported in the `temporally-inconsistent` experiment. The detection performance on the whole LitTempConsist is much lower due to the composition of this dataset. We remind that the whole `AndroZoo dataset` contains apps that span over two years (i.e., apps from 2019 and 2020). As for the whole `Literature dataset`, it contains Android apps that are spanning over eight years (i.e., apps from 2010 to 2018), which makes this dataset considerably difficult for all the classifiers.

In the experiments involving `Literature datasets`, DREBIN yielded the highest F1 score in nine out of ten experiments. DREBIN's feature set seems to be more suitable to detect the apps created before and until 2018, which is demonstrated by the `temporally-inconsistent` and the `temporally-consistent` experiments, respectively. Indeed, DREBIN has not outperformed the other detectors only on the whole `Literature dataset` but also on its subsets created in the sub-years of 2010-2018. As for the `AndroZoo datasets`, no approach has reported the highest detection performance in all the experiments. Consequently, no specific feature set from the evaluated state-of-the-art ap-

Table 4: The average F1 score reported by state-of-the-art approaches versus the combination of features versus the combination of classifiers

		Temporally inconsistent								**Temporally consistent**							
		Literature dataset					**AndroZoo dataset**			**Literature dataset**					**AndroZoo dataset**		
		Wh	**Dr**	**Rv**	**Mm**	**Ml**	**Wh**	**19**	**20**	**Wh**	**Dr**	**Rv**	**Mm**	**Ml**	**Wh**	**19**	**20**
RQ1	**DREBIN**	**0.92**	**0.94**	**0.97**	**0.98**	**0.96**	0.96	0.96	0.97	**0.44**	**0.87**	**0.94**	**0.92**	0.86	0.85	0.94	0.82
	Reveal	0.68	0.44	0.9	0.94	0.89	0.95	0.95	0.95	0.38	0.41	0.81	0.83	0.62	0.89	0.94	0.85
	MaMaF	0.48	0.31	0.9	0.94	0.82	0.95	0.95	0.97	0.19	0.32	0.88	0.89	0.64	0.92	**0.98**	0.86
	MaMaP	0.71	0.48	0.94	0.95	0.95	0.96	**0.97**	**0.98**	0.22	0.14	0.94	0.85	0.74	0.92	0.98	**0.87**
	MalD	0.88	0.87	0.94	0.95	0.95	0.96	0.96	0.97	0.33	0.7	0.86	0.79	0.87	**0.93**	0.96	0.87
	MalH	0.89	0.89	0.95	0.96	0.96	0.97	0.96	0.97	0.33	0.7	0.89	0.83	**0.88**	0.93	0.96	0.87
	MalK	0.89	0.9	0.95	0.96	0.95	0.96	0.96	0.97	0.36	0.69	0.89	0.81	0.87	0.9	0.95	0.86
	MalCl	0.89	0.88	0.95	0.96	0.96	**0.97**	0.96	0.97	0.4	0.77	0.89	0.84	0.88	0.93	0.96	0.87
	MalA	0.89	0.89	0.95	0.96	0.96	0.96	0.95	0.97	0.34	0.71	0.89	0.83	0.87	0.91	0.96	0.87
	MalCo	0.89	0.89	0.95	0.96	0.96	0.96	0.96	0.97	0.34	0.7	0.89	0.83	0.87	0.91	0.96	0.87
RQ3	**LinearSVC**	0.78	0.83	0.94	0.93	0.89	0.87	0.85	0.87	0.48	0.73	0.87	0.81	0.71	0.81	0.86	0.76
	RF	0.91	0.92	0.97	0.97	0.97	0.98	0.98	0.98	0.38	0.85	0.79	0.77	0.86	0.93	0.98	0.87
	KNN	0.86	0.83	0.92	0.95	0.91	0.94	0.94	0.93	0.3	0.76	0.72	0.74	0.82	0.85	0.88	0.74
	AdaBoost	0.86	0.86	0.97	0.96	0.95	0.96	0.95	0.97	0.45	**0.91**	0.95	0.9	0.81	0.93	0.97	0.86
	Bagging	**0.94**	**0.96**	**0.98**	**0.97**	**0.97**	**0.98**	**0.98**	**0.99**	**0.49**	0.85	0.94	0.86	0.85	**0.94**	0.98	**0.87**
	GradBoosting	0.9	0.92	0.98	0.97	0.97	0.97	0.97	0.98	0.48	0.82	**0.96**	**0.9**	**0.88**	0.93	**0.98**	0.87
RQ4	**MajorVote**	0.92	0.92	0.96	0.97	0.97	0.97	0.97	0.98	0.33	0.85	0.91	0.85	0.87	0.93	0.97	0.87
	AvgProba	0.91	0.92	0.96	0.96	0.97	0.97	0.97	0.97	0.32	0.83	0.91	0.85	0.89	0.94	0.97	0.87
	AccWProba	0.91	0.92	0.96	0.96	0.97	0.97	0.97	0.97	0.32	0.84	0.91	0.85	0.89	0.94	0.97	0.87
	F1WProba	0.91	0.91	0.96	0.96	0.97	0.97	0.97	0.97	0.33	0.81	0.91	0.85	0.89	0.94	0.97	0.87
	MinProba	0.3	0.14	0.92	0.94	0.82	0.93	0.93	0.93	0.12	0.0	0.63	0.62	0.38	0.83	0.96	0.79
	MaxProba	0.83	0.81	0.86	0.93	0.86	0.94	0.94	0.95	0.4	0.54	0.95	**0.93**	0.85	0.89	0.89	0.86
	ProdProba	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	StaPredSVM	0.93	0.93	0.97	0.97	0.96	0.96	0.96	0.97	0.33	0.82	0.95	0.87	0.85	0.85	0.94	0.83
	StaProbSVM	**0.94**	**0.95**	0.97	0.97	0.98	0.97	0.97	0.98	0.39	0.84	0.92	0.84	**0.9**	0.89	0.95	0.85
	StaPredRF	0.92	0.93	0.97	0.97	0.97	0.97	0.97	0.97	0.34	0.79	0.94	0.88	0.85	0.85	0.94	0.83
	StaProbRF	0.94	0.95	0.97	0.97	0.97	0.97	0.97	0.98	0.38	0.8	0.94	0.89	0.89	0.9	0.95	0.85
	StaPredKNN	0.92	0.92	0.97	0.97	0.97	0.97	0.97	0.97	0.39	0.79	0.92	0.87	0.86	0.86	0.94	0.82
	StaProbKNN	0.92	0.94	0.96	0.97	0.97	0.97	0.97	0.98	0.42	0.8	0.92	0.83	0.9	0.9	0.94	0.87
	StaPredMLP	0.93	0.93	0.97	0.97	0.97	0.97	0.97	0.97	0.38	0.8	0.94	0.88	0.85	0.86	0.94	0.83
	StaProbMLP	0.94	0.95	0.97	0.97	0.97	0.97	0.97	0.98	0.39	0.81	0.95	0.85	0.89	0.87	0.95	0.85
	EnsemSelect	0.94	0.95	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	**0.98**	**0.43**	**0.89**	**0.96**	0.91	0.89	**0.94**	**0.98**	**0.88**

Wh: Whole dataset, **Dr**: DREBIN dataset, **Rv**: RevealDroid dataset, **Mm**: MaMaDroid dataset, **Ml**: MalScan dataset, **19**: 2019 dataset, **20**: 2020 dataset

The cells highlighted in grey show the best detector for each dataset and for each RQ based on the F1 score before rounding

* We note that we have verified the standard deviation of the F1 scores over the ten runs of the experiments, and our results showed that the F1 scores do not vary sensibly.

proaches consistently helps to detect the highest number of malware created between 2019 and 2020.

We also observe that no approach has reported the highest detection performance on all the datasets. Specifically, DREBIN has achieved the best F1 score in nine out of 16 experiments. MaMaDroid Package is considered the best approach in three experiments. As for MalScan Closeness, MalScan Degree, MalScan Harmonic, and MaMaDroid Family, each of them has

reported the highest F1 score on one dataset. In the seven experiments where DREBIN has not reported the highest detection performance, the difference in the F1 score between DREBIN and the best approach on each dataset varies from 1 to 8 percentage points.

We further conduct a statistical test to compare the F1 scores reported by the state-of-the-art classifiers in all the datasets. We rely on the non-parametric Friedman test (Friedman 1937) that is designed to compare multiple data groups. Our selection of this test is motivated by the fact that our dataset of F1 scores does not follow a normal distribution. Additionally, previous studies (Perinetti 2016; Parab and Bhalerao 2010; Sheldon et al. 1996) have recommended using the Friedman test to statistically compare more than two datasets. Furthermore, Demšar (2006) have examined several statistical tests to compare ML classifiers and advised to use the Friedman test when comparing multiple classifiers on multiple datasets.

The null hypothesis states that state-of-the-art malware detectors have statistically equivalent detection performance. The Friedman test has reported a p_value of 3.75^{-81}, which means that the null hypothesis can be rejected. This result shows that our classifiers do not have the same detection performance. To conduct a pairwise comparison on our classifiers, we proceed with the Nemenyi (Nemenyi 1963) Post-Hoc test. This test aims to identify which classifiers have different detection performances after the null hypothesis of the Friedman test is rejected. We represent the p-values for the different pairs of classifiers in the sub-figure (a) of Figure 1.

As shown in Figure 1 (a), many state-of-the-art malware detector pairs do not have the same detection performance. For example, DREBIN's performance is not similar to that of five classifiers, including MaMaDroid Package, which has outperformed it on six datasets with a maximum difference of *seven* percentage points. This result confirms our observations that state-of-the-art approaches do not perform equally in all the settings.

Overall, our results show that the performance of state-of-the-art Android malware detectors is highly affected by the dataset used for the evaluation. Also, none of the studied approaches has reported the best detection performance in all the settings. Such finding motivates to further analyse the similarities and differences of state-of-the-art approaches by examining the malware they detect.

RQ1 answer: No state-of-the-art Android malware detector consistently outperforms the others.

3.2 **RQ2:** To what extent do state-of-the-art approaches detect similar/different malware?

In this section, we propose to examine the type of malware detected by state-of-the-art approaches in order to inspect their similarities/differences and

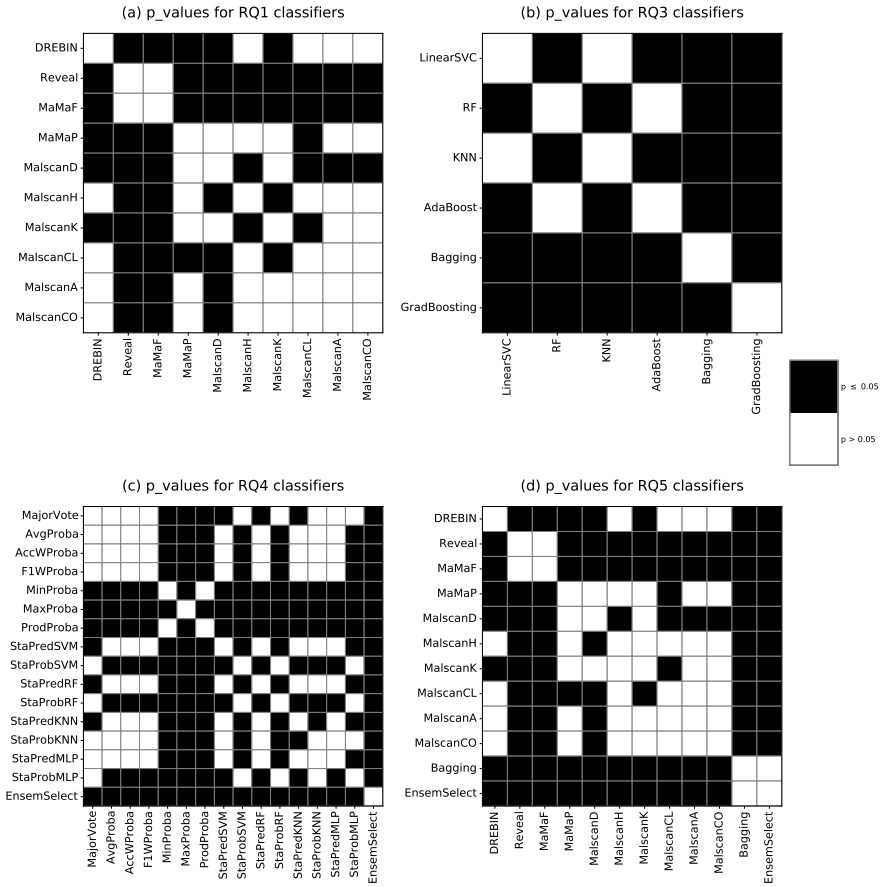


Fig. 1: The p_values of the Nemenyi pairwise comparison of all the classifiers

assess whether some classifiers perform better at detecting specific malware families. To this end, we first collect the detection reports for malware samples from VirusTotal[5]. Then, we leverage AVCLASS (Sebastián et al. 2016) to process the detection reports and assign a unique family label to each malware app. We infer the family label for malware apps in both the whole `Literature dataset` and the whole `AndroZoo dataset`. Overall, 642 and 204 unique malware families are present in `Literature dataset` and `AndroZoo dataset` respectively.

We start our investigation by identifying the family labels present in the test sets. Specifically, since we repeat the experiments ten times, we gather the family labels from the ten test subsets. Then, we merge these family labels to identify the top families in the test sets on average. For each top family, we

[5] `www.virustotal.com`

investigate how many samples belonging to that family are correctly detected by our approaches in the ten test splits on average.

We conduct our experiments on the whole `Literature dataset` and the whole `AndroZoo dataset` in both `Temporally consistent` and `Temporally inconsistent` settings. We select four top families from each setting and we present them in Table 5. We also report the results for the top 20 families in each setting in Table 11 and Table 12 in Appendix.

Table 5: Proportion of malware samples detected by state-of-the-art approaches and belonging to four top families

	Families	#	DREBIN	Reveal	MaMaF	MaMaP	MalD	MalH	MalK	MalCl	MalA	MalCo
LitTempInconsist	dowgin	352	**98.6**	68.8	63.1	86.9	93.2	94.0	94.3	94.0	94.0	94.0
	airpush	214	86.4	57.0	6.1	72.4	87.4	**91.6**	90.2	91.1	**91.6**	90.7
	adwo	195	82.6	72.8	11.8	62.6	85.6	86.2	**86.7**	85.6	86.2	86.2
	youmi	111	**82.9**	66.7	38.7	51.4	82.0	**82.9**	85.6	**82.9**	**82.9**	**82.9**
LitTempConsist	jiagu	408	**77.2**	64.0	0.0	0.2	0.7	10.3	0.7	0.7	19.9	19.9
	dnotua	303	5.0	5.0	0.3	11.2	**94.4**	2.6	94.1	94.1	2.6	2.6
	smsreg	136	**94.1**	77.9	36.0	52.2	57.4	66.9	63.2	63.2	69.1	69.1
	secapk	122	40.2	**92.6**	4.9	4.9	41.8	43.4	43.4	43.4	43.4	43.4
AndTempInconsist	secneo	69	84.1	**98.6**	0	**98.6**	79.7	58.0	79.7	79.7	78.3	78.3
	ewind	8	**100.0**	62.5	75.0	**100.0**	75.0	75.0	87.5	75.0	75.0	75.0
	datacollector	7	**100.0**	85.7	0.0	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	kuguo	6	**83.3**	**83.3**	0.0	66.7	66.7	66.7	66.7	66.7	66.7	66.7
AndTempConsist	hiddad	32	0.0	3.1	0	3.1	15.6	**21.9**	15.6	**21.9**	**21.9**	**21.9**
	joker	11	27.3	0	0	0	63.6	63.6	63.6	**72.7**	63.6	63.6
	emagsoftware	9	**66.7**	33.3	0	0	11.1	11.1	11.1	11.1	11.1	11.1
	autoins	7	**100.0**	**100.0**	0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

The entries in bold show the best detector for each malware family

We observe that some state-of-the-art approaches detect some families in similar proportions. For instance, DREBIN and some MalScan variants detect the same number of malware from `youmi` family in the LitTempInconsist setting. Similarly, RevealDroid and MaMaDroid Package detect the same proportion of apps from `secneo` family in the AndTempInconsist setting. Besides, compared to the other techniques, some approaches seem more efficient at detecting specific families. For example, the `secapk` family is effectively detected by RevealDroid in the LitTempConsist setting. In AndTempConsist, DREBIN is the approach that detects the highest proportion of malware from `emagsoftware` family.

Our results show that some state-of-the-art approaches share similarities since they detect specific families in similar proportions. Moreover, our classifiers also exhibit differences in their detected malware. Specifically, some approaches are more efficient than others at detecting some malware families. These insights combined with the finding from the previous RQ motivate to

combine knowledge from the state-of-the-art approaches (i.e., feature sets or predictions) to improve or stabilise the detection performance.

RQ2 answer: Some families are detected very accurately by some state-of-the-art approaches but almost completely escape the detection of some other approaches. The vast majority of families are **not** accurately detected by all the approaches.

3.3 **RQ3:** Does merging the *feature sets* from state-of-the-art approaches lead to a high-performing malware detector in all the settings?

In this section, we investigate whether merging features from Android malware detectors has an added value on the detection performance. Specifically, we aim to assess whether such a method can lead to high detection scores independently of the dataset. In Section 3.1, we have considered an approach as a whole (i.e., feature set + algorithm). In this experiment, we consider only the set of features proposed by each detector, and we merge them to create a single set of features. This latter is then used to train an ML algorithm to construct a new malware detector. Since the studied state-of-the-art approaches rely on different ML algorithms, we train the merged feature set using the same algorithms. Thus, we construct three malware detectors using LinearSVC, Random Forest, and K-Nearest Neighbour, and we train them with the merged feature set. Moreover, we also assess the detection performance of three additional ML algorithms:

- `AdaBoost` (Freund and Schapire 1997), which fits a series of base classifiers (e.g., Decision Tree) on the dataset such that each classifier focuses more on the incorrect predictions made by the previous classifier. This method assigns higher weights to the incorrectly predicted samples in order to enhance their prediction by the subsequent classifiers.
- `Bagging` (Breiman 1996), which trains a series of base classifiers on random subsets of the dataset and aggregates their predictions.
- `GradientBoosting` (Friedman 2001), which fits a series of base classifiers on the dataset in order to improve the prediction performance. Each classifier is trained to minimise the prediction errors of the previous classifier using the Gradient descent algorithm.

We again conduct our experiments under various experimental setups similarly to RQ1 (cf. Section 3.1). We report the F1 scores of our experiments in the middle part of Table 4. We also present the Recall, Precision, and Accuracy values of our evaluation in Table 7, Table 8, Table 9, and Table 10 in the Appendix.

We observe that the six malware detectors that are trained with the merged feature set also report detection performance that vary across datasets. Specifically, the whole LitTempConsist is still considered as the most difficult dataset since the highest F1 score reported by these classifiers is 0.49. On the 2020 AndTempInconsist, the highest F1 score has reached a value of 0.99.

Compared to the other algorithms trained on the merged feature set, `Bagging` reports the highest F1 score in 11 out of 16 experiments. For `Gradient-Boosting` and `AdaBoost`, they achieve the best detection score in four and one experiment respectively. In the five experiments where `Bagging` has not reported the highest detection scores, the difference in the F1 score between this detector and the best approaches reaches a maximum value of six percentage points.

We also compare the detection performance of these classifiers using the Friedman test. The p_value of the test is 8.85^{-111}, which indicates that the classifiers trained on the merged feature set do not have the same detection performance. We conduct the Nemenyi test and report its results in the subfigure (b) of Figure 1.

We observe that `Bagging` has a different detection performance than that of the other classifiers including those that have outperformed it on five datasets. Overall, our results show that none of the classifiers trained on the merged feature set has reported the highest detection performance on all the datasets.

RQ3 answer: Merging features from state-of-the-art malware detectors does not produce a high performing classifier in *all* the settings.

3.4 **RQ4:** Does combining *predictions* from state-of-the-art approaches lead to a high-performing malware detector in all the settings?

Following up on the findings of RQ3, we hypothesise that the performance of state-of-the-art approaches is brought by the right association between feature sets and learning algorithms. Therefore, we investigate the possibility to exploit the combined value of detectors via combining their independent predictions. To that end, we consider Ensemble Learning and study its impact on the detection performance. In our experiment, we consider the detectors trained in RQ1 (cf. Section 3.1) as base learners for Ensemble Learning, and we examine whether the combination of their predictions produces a high-performing malware detector on all the datasets.

Among the many ways of combining model predictions, which are commonly referred to as Ensemble Learning in the literature (Sagi and Rokach 2018; Dong et al. 2020), we consider the following cases:

- `Majority Voting`, where an app is considered as malware if it is detected by the majority of the classifiers (i.e., in our case at least 6 out of the 10 classifiers). Otherwise it is predicted as benign.
- `Average Probability`, which represents the average of the probability [6] scores given by the ten classifiers in the prediction of maliciousness. An app is predicted as malware if this Average probability is over 0.5.

[6] The probability of prediction is a score returned by the classifiers. It is between 0 and 1

- `Accuracy Weighted Probability`, where the probabilities of each classifier are weighted according to their Accuracy metric. An app is predicted as malware if the weighted Probability for malware class is higher than the weighted Probability for benign class.
- `F1 Weighted Probability`, where the probabilities of each classifier are weighted according to their F1 metric. An app is predicted as malware if the weighted Probability for malware class is higher than the weighted Probability for benign class.
- `Min Probability`, which represents the minimum score among the probability scores given by the ten classifiers. An app is predicted as malware if this Min probability is over 0.5.
- `Max Probability`, which represents the maximum score among the probability scores given by the ten classifiers. An app is predicted as malware if this Max probability is over 0.5.
- `Product Probability`, which represents the product of the probability scores given by the ten classifiers in the prediction of maliciousness. An app is predicted as malware if this Product Probability is over 0.5.
- `Stacking Prediction` (Wolpert 1992), where the predictions of each classifier are used to train a binary meta-classifier. We evaluated the Stacking method using four meta-classifiers: SVM, RF, KNN, and Multi-Layer Perception (MLP), with three hidden layers of 32, 64, and 128 neurons, respectively. The final predictions of this method are given by the meta-classifier.
- `Stacking Probability`, where the prediction probabilities of each classifier are used to train a binary meta-classifier. Similarly, we evaluated Stacking Probability using four meta-classifiers: SVM, RF, KNN, and Multi-Layer Perception (MLP), with the same architecture as in `Stacking Prediction`.
- `Ensemble Selection`, where the probabilities of each classifier are weighted according to its overall performance on specific metrics (i.e., F1-score, Recall, ...). Since such a performance must be determined beforehand, we use a validation dataset that serves to iteratively[7] infer the weights for each classifier (Caruana et al. 2004).

Experimental results with the described Ensemble Learning techniques are provided in the lower part of Table 4. Again, we provide the detailed scores in Table 7, Table 8, Table 9, and Table 10 in the Appendix.

We observe that the detection performance still varies significantly across datasets: `Min Probability` sees the most important variation of 84 percentage points (0.12 for the whole LitTempConsist to 0.96 for `2019` AndTempConsist). The difficulty of the whole LitTempConsist is also confirmed when combining the predictions since the highest F1 score reported in that dataset is 0.43.

The evaluation results for the 16 prediction combination methods show that none of these methods has reported the highest F1 score on all the

[7] In our experiment, we fix the number of iterations to 5000

datasets. Specifically, `Ensemble Selection` is the best technique in 12 experiments. `Stacking Probability` with `SVM` achieves the highest F1 score in three experiments. As for `Max Probabilities`, it outperformed the others on one dataset. When `Ensemble Selection` is not the highest performing classifier, the difference in F1 score between `Ensemble Selection` and the best method is at most two percentage points.

We again conduct the Friedman test to compare the detection performance of the Ensemble Learning classifiers. The test reports a p_value of 1.42^{-249}, which confirms that these classifiers do not have the same detection performance. We then proceed with the Nemenyi test and report its results in the sub-figure (c) of Figure 1.

The sub-figure shows that the classifiers used to combine the predictions do not perform similarly. Specifically, the detection performance of `Ensemble Selection` is not similar to that of all the evaluated classifiers, including `Max Probabilities` and `Stacking Probability` with `SVM`, which have outperformed it on four datasets. Our results show that none of the Ensemble Learning classifiers has yielded the highest detection performance on all the datasets.

RQ4 answer: Combining predictions from state-of-the-art approaches does not produce a classifier that outperforms the others on *all* the datasets.

3.5 **RQ5:** Does combining *feature sets* or *predictions* from state-of-the-art approaches lead to classifiers that significantly outperform the original detectors?

In this section, we aim to compare the detection performance of the state-of-the-art classifiers and the best methods to combine the features and the predictions. The evaluation of the state-of-the-art malware detectors (c.f., Section 3.1) showed that no approach has outperformed the others on all the datasets. For example, DREBIN has reported the highest F1 score in nine out of 16 experiments, but other approaches have remarkably outperformed it on the `AndroZoo datasets`. In Section 3.3, we have assessed the added value of the merged feature set using six classifiers. Our results showed that `Bagging` achieved the highest F1 score in 11 out of 16 experiments. On the DREBIN LitTempConsist dataset, `AdaBoost` has outperformed `Bagging` with 6 percentage points. With the combination of predictions experiments, we have observed the same pattern: No Ensemble Learning method has reported the highest F1 score in all the settings. For example, `Ensemble Selection` achieved the best detection scores in 12 experiments, but other methods have outperformed it in four evaluation experiments. However, the difference in F1 score between `Ensemble Selection` and the best approaches in these four experiments is at most two percentage points.

Before proceeding with the statistical test, we first compare the detection scores of the best state-of-the-art malware detectors with those reported by the combination methods in RQ3 and RQ4. Specifically, we select from each RQ the combination method that has most often outperformed the others. For RQ3, we select `Bagging` as the best classifier trained with the merged feature set. As for RQ4, `Ensemble Selection` is considered the best method to combine the predictions. We refer to Table 4 to compare the detection performance of these two methods with that of the best state-of-the-art classifiers on each dataset.

Overall, `Bagging` has increased the detection performance in nine experiments. The increase in the F1 score is at most two percentage points except for the whole LitTempConsist where it has reached five percentage points. This classifier has also decreased the F1 score in four experiments by one, two, six, and one percentage point, respectively. In the remaining three experiments, `Bagging` has reported the same detection performance as the best state-of-the-art approaches. As for `Ensemble Selection`, it has increased the detection performance by at most two percentage points in 11 experiments. This method has also reported the same detection performance as the best approaches in three experiments and decreased the F1 score by one percentage point in two experiments.

Neither `Bagging` nor `Ensemble Selection` has remarkably increased the detection performance of state-of-the-art malware detectors. While it has enhanced the F1 score by five percentage points on one dataset, `Bagging` has also decreased the F1 score by six percentage points on one dataset. For `Ensemble Selection`, despite improving the F1 score in 11 experiments, this improvement is at most two percentage points. Nevertheless, `Ensemble Selection` has generally maintained the highest detection performance of state-of-the-art malware detectors independently of the dataset since it has maintained the least performance gap with the best classifiers on all the datasets.

To validate our observations, we conduct the Friedman test on the F1 scores reported by the state-of-the-art classifiers, `Bagging` and `Ensemble Selection`. Since the p_value of the test is 5.42^{-179}, we conduct the Nemenyi test and report our results in the sub-figure (d) of Figure 1.

As shown in the sub-figure, the detection performance of both `Bagging` and `Ensemble Selection` is different than that of the state-of-the-art classifiers. Moreover, the p_value of the test that compares `Bagging` and `Ensemble Selection` is greater than 0.5, which means that we failed to reject the null hypothesis. Our results suggest that there is insufficient evidence to affirm that the detection performance of these two classifiers is different.

To sum up, both `Bagging` and `Ensemble Selection` have generally maintained the highest detection performance of the state-of-the-art approaches independently of the datasets.

RQ5 answer: Despite not improving the detection performance overall, combining the features or the predictions from state-of-the-art approaches (with `Bagging` and `Ensemble Selection` respectively) produces classifiers that maintain the best detection scores independently of the experimental datasets.

4 Discussion

The literature of Android malware detection lavishes with a huge number of malware classifiers. Each approach aims to capture malware samples by proposing a set of features that is compiled to approximately represent app behaviour. In this study, we consider state-of-the-art approaches published in top venues, and we perform an independent evaluation of their performance. Our evaluation dataset includes a diverse set of apps, spanning across a decade (2010-2020) of app development. Our aim is to challenge the classifiers with diverse samples. We further executed experiments where dataset selections are `temporally-consistent` (in contrast with typical random sampling), in order to assess malware classifiers' ability to cope with emerging malware. Overall, considering all experimental scenarios, the results show that none of the studied approaches stands out across all settings.

In this section, we discuss an important insight from our study: while combining different approaches does not systematically improve the achievable performance, we note that it can help maintain a high performance across all settings.

4.1 Ensuring high detection performance across datasets

Our study shows that malware detectors have significant variability in performance from one dataset to another. Furthermore, no state-of-the-art malware detector could outperform all others in all settings. These results raise questions about the characterisation of the added-value of each studied approach as well as its suitability for deployment in production.

In an attempt to build a malware classifier that exploits the added-value of all studied classifiers, we have investigated two main approaches: merge of all feature sets and combination of classifiers' predictions. Our experiments show that `Bagging` and `Ensemble Selection` have reported promising results: the yielded classifier generally achieves, in all scenarios, a detection score that is as good as the best score reported by individual approaches. Therefore, these combination methods ensure that the highest detection performance is stabilised independently of the dataset.

Overall, the observed results further stress the need to consider large-scale and diverse datasets to limit the biases when evaluating Android malware classification approaches.

4.2 Hypothetic reasons behind the failure of Ensemble Learning to outperform the state of the art

Generally, Ensemble Learning methods aim to enhance the detection performance of the base learners. In our study, however, these methods did not help to outperform the state of the art, although they have provided a detection performance stability across datasets. In the best case, `Bagging` and `Ensemble Selection` methods have increased the highest F1 score reported by the base learners by five and two percentage points, respectively.

From Table 4, we observe that there is still room to improve the state of the art, in particular when the experiments are performed in a `temporally-consistent` manner. Yet, our experiments show that combining feature sets or predictions from these state-of-the-art classifiers does not lead to the hoped improvement. Below we enumerate potential reasons why combining the state of the art has not led to a classifier that surpasses all individual approaches:
① **There is a significant overlap of false-negatives in state-of-the-art classifications:** We hypothesise that combining state-of-the-art approaches could not enhance the detection performance due to the presence of malware apps that are actually "difficult to detect" for all the approaches. Specifically, the malware that has escaped the detection of the best approaches could not be detected by the other approaches either. To verify our hypothesis, we examine the pairwise overlap of False Negatives (FNs) for the best detector and each of the other detectors considered in RQ1 on average. We provide in Figure 2, the distribution of the FNs overlap for the whole datasets. We also present in Table 6 the number of FNs overlap for the best detector and each of the other classifiers.

Table 6: The average number of overlapping FNs for the best detector and each of the other detectors on the whole datasets

	LitTempInconsist	LitTempConsist	AndTempInconsist	AndTempConsist
Best Approach	**DREBIN**	**DREBIN**	**MalCl**	**MalD**
DREBIN	-	-	148	625
Reveal	289	1127	190	659
MaMaF	356	1388	248	709
MaMaP	289	1262	192	688
MalD	201	512	237	-
MalH	200	1075	234	648
MalK	193	653	227	699
MalCl	200	548	-	679
MalA	200	1063	225	661
MalCo	199	1063	229	658

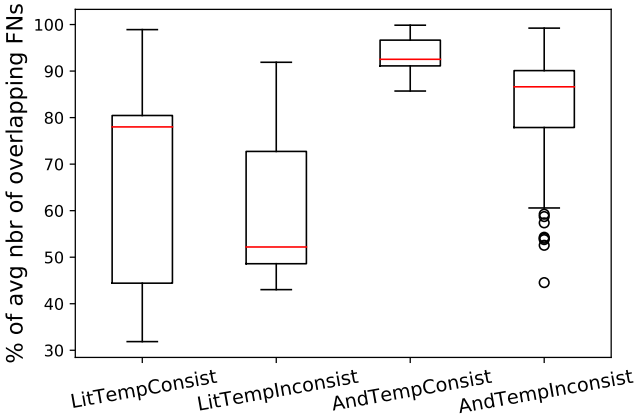


Fig. 2: The distribution of the average number of overlapping FNs for the best detector and each of the other approaches in the whole datasets

We observe that there is a significant overlap of FNs in all the datasets. This overlap ranges from 32% in LitTempConsist to 100% in AndTempConsist. These results suggest that malware apps that are "difficult to detect" for the best detector, in a given scenario, are also challenging for the other classifiers: such apps therefore escape the detection despite Ensemble Learning.

We also inspect the families of the overlapping FNs from the first data splits in order to identify the major families that are difficult to detect. Specifically, we identify the top five families of the overlapping FNs for the best approach and each detector in the whole datasets. We represent the distribution of the number of malware in the top families in Figure 3.

The results in Figure 3 show that many overlapping FNs do not belong to a known family (i.e., "SINGLETON"). We note that the label "SINGLETON", generated by AVCLASS, groups the malware that could not be attributed to any family. The overlap of these samples exceeds 500 malware, which shows that they are indeed difficult to detect. Regarding the known families, we observe that "fakeapp", "jiagu", "secapk", and "dnotua" represent the families with the highest number of overlapping FNs. Overall, inspecting how these samples differ from the other TPs belonging to the same families might help design new approaches that can detect the escaped malware.

② `Temporally-consistent` **experiments are challenging:**

In Table 4, we observe that the best combination results are achieved in the `temporally-inconsistent` experiments. Moreover, we have seen in Section 3.5 that `Ensemble Selection` has decreased the detection performance of the original classifiers in two experiments which are both `temporally-consistent`. Furthermore, the individual approaches themselves have somehow reported a poor detection performance especially for the Literature datasets (cf. Section 3.1). For the whole LitTempConsist, we have shown that all

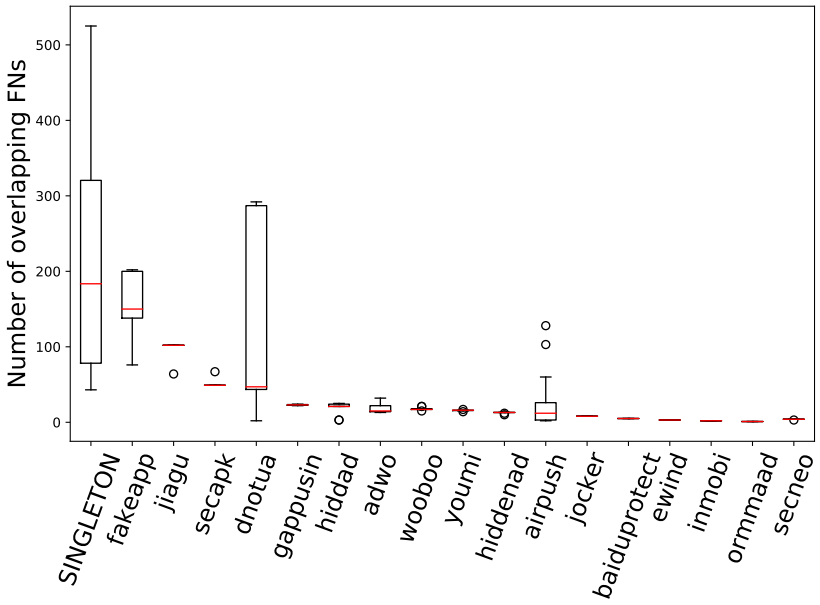


Fig. 3: The distribution of the number of overlapping FNs' from the top five families in the whole datasets

the state-of-the-art classifiers report F1 scores that are below 0.5. Since these classifiers make mistakes more often, their combination could not offer any improvements, and has even resulted in a slight decrease of the performance. We note that our results are in line with previous studies (Pendlebury et al. 2019; Allix et al. 2015) in which the authors show that the performance of malware detectors is significantly decreased in a `temporally-consistent` setting.

The limited performance reported on the `temporally-consistent` experiments can be explained by the evolution of Android malware and the emergence of new malware families. Indeed, Android malware is evolving fast, and new families can exhibit previously-unknown behaviours. In the `temporally-consistent` experiments, the test dataset is likely to contain malware belonging to families that were unseen in the training dataset. In the `temporally-inconsistent` experiments, this situation is possible, but less likely due to the randomness of the split. Given that the training is supposed to characterise maliciousness, if the training set is not representative of the different families, the model will not generalise to samples in the test set, which would lead to poor detection performance.

③ **The diversity of the feature sets is limited:** The 10 studied detector variants each leverage a different feature set. In the whole AndTempInconsist dataset, we have found that the overall number of features across all studied approaches surpasses 19 Million features. This huge number of features could have suggested that, altogether, state-of-the-art approaches have

sufficient information to correctly predict malware apps. Unfortunately, our experiments show that even merging all the features set to train a single machine learning algorithm does not lead to capturing all malware samples. We thus hypothesise that the overall feature set is not more representative (i.e., does dot capture more relevant information) than individual feature sets proposed by different approaches. It is indeed plausible that the different feature sets are actually redundant across approaches, with respect to malicious behaviour characterisation. This raises a concern in the literature on the added-value of ever-renewed feature sets using the same types of analyses.

Recently, researchers have started to investigate novel ways to represent Android apps (Daoudi et al. 2021c; Sun et al. 2021; Huang and Kao 2018; Ding et al. 2020). In particular, the feature engineering process, which was largely manual, has been tasked to be resolved via deep learning. To that end, artefacts from the app package (e.g., the DEX file, Manifest, etc.) can be processed (e.g., via image representation) to be fed to neural networks for automatic features extraction. Such alternative features may help improve malicious behaviour characterisation.

④ **Classification algorithms leveraged in our study may have limited capabilities:** The studied approaches rely on three common classification algorithms: Linear SVM, RF, and KNN. We further relied on these same algorithms and three others to train classifiers with merged feature sets (c.f., Section 3.3), resulting in limited performance improvement. We hypothesise that these algorithms may be unsuitable for individually processing the variety of feature types (e.g., Permissions, the representation of the apps call graphs as social networks, as Markov Chains, ...), leading to poor detection performance improvement.

⑤ **The combination methods may not be suitable:** We have investigated the impact of combining state-of-the-art malware detectors using 16 Ensemble Learning methods. While these combination methods have, at best, maintained the highest detection performance of the base learners, they could not generally help to catch the escaped malware. We hypothesise that we may have not been able to identify a relevant combination method for leveraging and enhancing the power of each approach when used in conjunction with others. More sophisticated Ensemble Learning techniques may lead to different results in future work.

⑥ **AV labels used in our study may be noisy (Hurier et al. 2016; Salem et al. 2021; Xu et al. 2021):** Android malware datasets are generally created using labels from AV engines or online services such as VirusTotal[8]. Antivirus engines have been used to label most of the apps in our datasets as malware or benign. Since the AV engines may have different classification decisions, their use can result in a noisy dataset.

Researchers usually consider an app as benign when it is not flagged by any antivirus. In order to label an app as malware, a threshold of antivirus agreements needs to be defined. Specifically, the malicious label is attributed

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

to the apps that are detected by a number of antivirus engines that is equal to or above the specified threshold. Some researchers choose a higher threshold value in order to increase the likelihood that the apps are truly malware. Other researchers prefer to decrease the threshold value so that the "grey" malware are also included in the dataset. While the second strategy can indeed help to learn from the "difficult" malware, it can results in including False Positive labels in the ground-truth dataset. Our experiments use datasets, from different sources, which are compiled using different strategies. This may have introduced noise that is challenging to estimate.

4.3 Threats-to-validity

The results and findings of our study are subject to some threats to validity. In this section, we enumerate these threats and explain how we attempted to alleviate their impact. First, since the generalisability of our conclusions highly depends on the evaluation dataset, we have considered two large datasets of Android apps to extend the validity of our findings. The literature dataset has been used to evaluate Android malware detectors in the literature, and it includes over 197K apps. We have also removed the duplicated apps in the whole literature dataset to avoid evaluation biases and comply with the recent recommendations about sample duplication (Zhao et al. 2021). As for the AndroZoo dataset, it contains over 265K samples that we have collected from AndroZoo (Allix et al. 2016b) repository. Moreover, apps in our datasets span from 2010 to 2020, which helps to thoroughly assess the performance of the studied approaches. Furthermore, we have also included the Literature and AndroZoo subsets in our evaluation to diversify our settings. Second, we study the possibility of combining state-of-the-art malware detectors. Since Android malware detection literature is prolific, selecting the evaluated subjects is not straightforward. To eliminate any selection bias, papers from 16 major venues in Software Engineering, Security, and Machine Learning have been considered. Third, the implementations of the evaluated approaches might also bias our results. To mitigate this threat, we have considered the approaches that have been reported to be reproducible in the literature (Daoudi et al. 2021b). We relied on reproducible malware detectors to ensure that our results are valid and reflect the detection performance of the original approaches. Finally, the validity of our findings might be affected by the methods used to combine the evaluated approaches. We have mitigated this threat by considering a total of 22 combination methods: six methods to combine the feature set and 16 methods for the predictions. We have also repeated our experimental evaluations ten times to mitigate potential overfitting. Moreover, we have conducted statistical tests to compare the detection performance of the evaluated classifiers in order to validate our findings.

5 Related Work

Our study relates to the research direction that has put special effort on evaluating and building on published work, which is presented in Section 5.1. We also review the use of Ensemble Learning in Android malware detection in Section5.2.

5.1 Assessment of Existing work

Researchers have started long ago to invest in reviewing existing work on malware detection. A survey (Rossow et al. 2012) has been conducted to assess the methodological rigour and prudence of 36 malware execution papers and has stressed the need for the community to ensure better handling of the datasets.

The use of the most recent training labels from VirusTotal has been shown to artificially inflate the detection performance of malware detectors (Miller et al. 2016). A temporal label consistency constraint has then been introduced to ensure that the training labels are temporally precedent to the evaluation samples.

Ten sources of biases have been identified based on the revision of 30 papers from top-tier security venues (Arp et al. 2020). These biases can affect the results reported in machine learning based computer and network security research. A set of recommendations that include data collection, labelling, model design, and learning have then been proposed to mitigate such pitfalls.

In Android malware detection, the evaluation results reported in the literature have been carefully scrutinised and have been shown to be affected by temporal and spatial biases (Pendlebury et al. 2019; Allix et al. 2015). For instance, Tesseract (Pendlebury et al. 2019) has demonstrated that the performance of DREBIN and MaMaDroid is highly affected by these two biases. Similarly, it has been demonstrated that the 10-fold cross-validation evaluation method can positively and artificially inflate the evaluation results (Allix et al. 2016a).

Recently, an in-depth study (Daoudi et al. 2021a) has been conducted on DREBIN to analyse its inner working beyond its detection scores.

5.2 Ensemble Learning for Android malware detection

Due to its promising results in several domains, Ensemble Learning methods have attracted the attention of researchers to develop techniques to curb the spread of Android malware. Existing work has explored the use of Ensemble Learning with some selected features and ML algorithms. To the best of our knowledge, we are the first to investigate the use of Ensemble Learning with state-of-the-art Android malware detectors.

Random Forest is an Ensemble Learning algorithm that trains a set of Decision Tree classifiers as base learners. The class that is predicted by most

of the base learners is selected as the decision output of the Random Forest (Breiman 2001). This algorithm has been used both as base learner (Zhao et al. 2018; Zhao et al. 2019; Dhalaria and Gandotra 2020) and as an Ensemble Learning method (Yerima et al. 2015; Alam and Vuong 2013; Zhang and Jin 2016) for Android malware detection. For example, a total of 179 static features that include API calls, (Linux/Android) commands, and Permissions are extracted and fed to a Random Forest Ensemble Learner (Yerima et al. 2015). This same approach is re-used by augmenting the features set with semantics-based features extracted from the sinks and sources flows (Zhang and Jin 2016). RF has been combined with KNN as base Learners to predict malware using sensitive API Calls from a small dataset of 1044 apps (Zhao et al. 2018). Probabilities of predictions from RF and KNN have been weighted with 0.6 and 0.4 respectively to form the final decision. One year later, the same approach was slightly modified by adding the Permissions to the features set (Zhao et al. 2019).

Other methods of Ensemble Learning have also been evaluated. Stacking refers to training a meta model on the predictions of other based learners. Logistic Regression has been used as a Stacking algorithm to combine the output of Random Forest, SVM, and KNN algorithms that are trained using features from AndroMD dataset[9] (Dhalaria and Gandotra 2020). MuViDA (Appice et al. 2020) is a multi-view malware detection approach that is based on clustering followed by Stacking using Random Forest algorithm. SEDMDroid is a Stacking Ensemble method that relies on Multi-Layer Perceptrons as base learners and four types of features: Permissions, permission-rate, monitoring system events sensitive APIs, and data flow information (Zhu et al. 2020). Stacking has also been used with neural networks base learners and Dalvik instructions to predict malware (Zhang et al. 2015). Support Vector Machine algorithm has been used to assemble the prediction output of Naive Bayes classifiers (Palumbo et al. 2017). Mlifdect (Wang et al. 2017) is an Ensemble Learning approach that predicts an app as malware if the sum of probabilities of its base learners is above a predefined threshold.

Assembling the prediction of the base learners has also been investigated using Average, Maximum, Product of probabilities, and Majority Vote with features that include permissions, Standard OS and Android commands, and API-related features (Yerima et al. 2014). Majority Voting has been leveraged to assemble classifiers trained with permission features (Christianah et al. 2020), and with the combination of permissions and source code features (Milosevic et al. 2017). Soft voting has been used to combine the output of a Decision Tree, a Deep Neural Network, and an LSTM classifier that are trained using API calls, API frequency, and API sequence features. Another study has used Genetic algorithms to select Deep Belief Neural Networks base learners that have their predictions assembled using the majority voting (Wang et al. 2020).

[9] `https://www.kaggle.com/meghnadhalaria/andromd`

6 Conclusion

The literature of Android malware detection is prolific. Nevertheless, the expectation gap between the promising research results and the severe spread of malware suggests that our community needs to revisit the evaluation of the promise of state-of-the-art approaches. In this work, we contribute with a large-scale evaluation of four state-of-the-art malware detectors published at major venues, using two datasets of over 197k and 265k apps. We confirm previous results in the literature, which found that the performance of malware detectors is highly dependent on the dataset used in the evaluation. Particularly, no approach has reported the best detection results on all the settings, which casts doubts on the usability and the validity of the studied approaches in real-world settings.

In an attempt to stabilise the detection performance across all datasets, we have investigated the use of Ensemble Learning methods. Our results show that `Bagging` and `Ensemble Selection` methods are promising and can generally maintain the best detection scores independently of the dataset. To further facilitate future studies, we make available to the research community the extracted features (for 462k apps) following the approaches of ten detector variants.

7 Data availability

The datasets used in the present study are available in our repository:
`https://github.com/Trustworthy-Software/Combination-malware-detectors`

8 Acknowledgements

This work was partially supported (a) by the Fonds National de la Recherche (FNR), Luxembourg, under project CHARACTERIZE C17/IS/11693861, (b) by the University of Luxembourg under the HitDroid grant, (c) by the SPARTA project, which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 830892, and (d) by the Luxembourg Ministry of Foreign and European Affairs through their Digital4Development (D4D) portfolio under project LuxWAyS.

9 Conflict of interest

The authors declare that they have no conflict of interest.

References

Afonso VM, de Amorim MF, Grégio ARA, Junquera GB, de Geus PL (2015) Identifying android malware using dynamically obtained features. Journal of Computer Virology and Hacking Techniques 11(1):9–17

Alam MS, Vuong ST (2013) Random forest classification for detecting android malware. In: 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, pp 663–669, DOI 10.1109/GreenCom-iThings-CPSCom.2013.122

Allix K, Bissyandé TF, Klein J, LeTraon Y (2015) Are your training datasets yet relevant? In: Piessens F, Caballero J, Bielova N (eds) Engineering Secure Software and Systems, Springer International Publishing, Cham, pp 51–67, URL https://doi.org/10.1007/978-3-319-15618-7_5

Allix K, Bissyandé TF, Jérome Q, Klein J, State R, Le Traon Y (2016a) Empirical assessment of machine learning-based malware detectors for android. Empirical Software Engineering 21(1):183–211, DOI 10.1007/s10664-014-9352-6, URL https://doi.org/10.1007/s10664-014-9352-6

Allix K, Bissyandé TF, Klein J, Le Traon Y (2016b) Androzoo: Collecting millions of android apps for the research community. In: Proceedings of the 13th International Conference on Mining Software Repositories, ACM, New York, NY, USA, MSR '16, pp 468–471, DOI 10.1145/2901739.2903508, URL http://doi.acm.org/10.1145/2901739.2903508

Appice A, Andresini G, Malerba D (2020) Clustering-aided multi-view classification: A case study on android malware detection. Journal of Intelligent Information Systems 55(1):1–26

Arp D, Spreitzenbarth M, Hübner M, Gascon H, Rieck K (2014) Drebin: Efficient and explainable detection of android malware in your pocket. In: Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS), San Diego, CA

Arp D, Quiring E, Pendlebury F, Warnecke A, Pierazzi F, Wressnegger C, Cavallaro L, Rieck K (2020) Dos and don'ts of machine learning in computer security. arXiv preprint arXiv:201009470

Breiman L (1996) Bagging predictors. Machine learning 24(2):123–140

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

Brown G (2010) Ensemble learning. Encyclopedia of machine learning 312:15–19

Caruana R, Niculescu-Mizil A, Crew G, Ksikes A (2004) Ensemble selection from libraries of models. In: Proceedings of the Twenty-First International Conference on Machine Learning, Association for Computing Machinery, New York, NY, USA, ICML '04, p 18, DOI 10.1145/1015330.1015432, URL https://doi.org/10.1145/1015330.1015432

Christianah A, Gyunka B, Oluwatobi A (2020) Optimizing android malware detection via ensemble learning, URL https://www.learntechlib.org/p/217826

Daoudi N, Allix K, Bissyandé TF, Klein J (2021a) A deep dive inside drebin: An explorative analysis beyond android malware detection scores. ACM Transactions on Privacy and Security (TOPS) To appear

Daoudi N, Allix K, Bissyandé TF, Klein J (2021b) Lessons learnt on reproducibility in machine learning based android malware detection. Empirical Software Engineering 26(4):1–53, DOI 10.1007/s10664-021-09955-7, URL https://doi.org/10.1007/s10664-021-09955-7

Daoudi N, Samhi J, Kabore AK, Allix K, Bissyandé TF, Klein J (2021c) Dexray: A simple, yet effective deep learning approach to android malware detection based on image representation of bytecode. In: Wang G, Ciptadi A, Ahmadzadeh A (eds) Deployable Machine Learning for Security Defense, Springer International Publishing, Cham, pp 81–106, DOI 10.1007/978-3-030-87839-9\{_}4, URL https://doi.org/10.1007/978-3-030-87839-9_4

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. The Journal of Machine learning research 7:1–30

Dhalaria M, Gandotra E (2020) Android malware detection using chi-square feature selection and ensemble learning method. In: 2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC), pp 36–41, DOI 10.1109/PDGC50313.2020.9315818

Ding Y, Zhang X, Hu J, Xu W (2020) Android malware detection method based on bytecode image. Journal of Ambient Intelligence and Humanized Computing pp 1–10

Dong X, Yu Z, Cao W, Shi Y, Ma Q (2020) A survey on ensemble learning. Frontiers of Computer Science 14(2):241–258

Fereidooni H, Conti M, Yao D, Sperduti A (2016) Anastasia: Android malware detection using static analysis of applications. In: 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp 1–5, DOI 10.1109/NTMS.2016.7792435

Freund Y, Schapire RE (1997) A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences 55(1):119–139, DOI https://doi.org/10.1006/jcss.1997.1504, URL https://www.sciencedirect.com/science/article/pii/S002200009791504X

Friedman JH (2001) Greedy function approximation: a gradient boosting machine. Annals of statistics pp 1189–1232

Friedman M (1937) The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association 32(200):675–701, DOI 10.1080/01621459.1937.10503522, URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1937.10503522, https://www.tandfonline.com/doi/pdf/10.1080/01621459.1937.10503522

G DATA (2020) G DATA mobile malware report. URL https://www.gdatasoftware.com/news/1970/01/-36401-g-data-mobile-malware-report-harmful-android-apps-every-eight-seconds, accessed June 10, 2021

Garcia J, Hammad M, Malek S (2018) Lightweight, obfuscation-resilient detection and family identification of android malware. ACM Trans Softw Eng Methodol 26(3), DOI 10.1145/3162625, URL https://doi.org/10.1145/3162625

Huang TH, Kao H (2018) R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections. In: 2018 IEEE International Conference on Big Data (Big Data), pp 2633–2642, DOI 10.1109/BigData.2018.8622324

Hurier M, Allix K, Bissyandé TF, Klein J, Le Traon Y (2016) On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware. In: Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment - Volume 9721, Springer-Verlag, Berlin, Heidelberg, DIMVA 2016, p 142–162, DOI 10.1007/978-3-319-40667-1_8, URL https://doi.org/10.1007/978-3-319-40667-1_8

Idrees F, Rajarajan M, Conti M, Chen TM, Rahulamathavan Y (2017) Pindroid: A novel android malware detection system using ensemble learning methods. vol 68, pp 36–46, DOI https://doi.org/10.1016/j.cose.2017.03.011, URL https://www.sciencedirect.com/science/article/pii/S0167404817300640

Kaspersky (2021) Kaspersky security network. URL https://securelist.com/it-threat-evolution-q1-2021-mobile-statistics/102547/, accessed June 10, 2021

Mariconti E, Onwuzurike L, Andriotis P, De Cristofaro E, Ross G, Stringhini G (2017) MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In: ISOC Network and Distributed Systems Security Symposiym (NDSS), San Diego, CA

Miller B, Kantchelian A, Tschantz MC, Afroz S, Bachwani R, Faizullabhoy R, Huang L, Shankar V, Wu T, Yiu G, Joseph AD, Tygar JD (2016) Reviewer integration and performance measurement for malware detection. In: Caballero J, Zurutuza U, Rodríguez RJ (eds) Detection of Intrusions and Malware, and Vulnerability Assessment, Springer International Publishing, Cham, pp 122–141

Milosevic N, Dehghantanha A, Choo KKR (2017) Machine learning aided android malware classification. Computers & Electrical Engineering 61:266–274, DOI https://doi.org/10.1016/j.compeleceng.2017.02.013, URL https://www.sciencedirect.com/science/article/pii/S0045790617303087

Nemenyi PB (1963) Distribution-free multiple comparisons. Princeton University

Onwuzurike L, Mariconti E, Andriotis P, Cristofaro ED, Ross G, Stringhini G (2019) Mamadroid: Detecting android malware by building markov chains of behavioral models (extended version). ACM Trans Priv Secur 22(2):14:1–14:34, DOI 10.1145/3313391, URL http://doi.acm.org/10.1145/3313391

Palumbo P, Sayfullina L, Komashinskiy D, Eirola E, Karhunen J (2017) A pragmatic android malware detection procedure. Computers & Security 70:689–701, DOI https://doi.org/10.1016/j.cose.2017.07.013, URL https://www.sciencedirect.com/science/article/pii/S0167404817301542

Parab S, Bhalerao S (2010) Choosing statistical test. International journal of Ayurveda research 1(3):187

Pendlebury F, Pierazzi F, Jordaney R, Kinder J, Cavallaro L (2019) TESSERACT: Eliminating experimental bias in malware classification across space and time. In: 28th USENIX Security Symposium (USENIX Security 19), USENIX Association, Santa Clara, CA, pp 729–746, URL https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury

Perinetti G (2016) Statips part i: Choosing statistical test when dealing with differences. South European Journal of Orthodontics and Dentofacial Research 3(1):3–4

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

Sagi O, Rokach L (2018) Ensemble learning: A survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8(4):e1249

Salem A, Banescu S, Pretschner A (2021) Maat: Automatically analyzing virustotal for accurate labeling and effective malware detection. ACM Trans Priv Secur 24(4), DOI 10.1145/3465361, URL https://doi.org/10.1145/3465361

Sebastián M, Rivera R, Kotzias P, Caballero J (2016) Avclass: A tool for massive malware labeling. In: International Symposium on Research in Attacks, Intrusions, and Defenses, Springer, pp 230–253

Sheldon MR, Fillyaw MJ, Thompson WD (1996) The use and interpretation of the friedman test in the analysis of ordinal-scale data in repeated measures designs. Physiotherapy Research International 1(4):221–228

Sun T, Daoudi N, Allix K, Bissyandé TF (2021) Android malware detection: Looking beyond dalvik bytecode. In: Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering Workshops, ASE '21

Wang J, Jing Q, Gao J, Qiu X (2020) Sedroid: A robust android malware detector using selective ensemble learning. In: 2020 IEEE Wireless Communications and Networking Conference (WCNC), pp 1–5, DOI 10.1109/WCNC45663.2020.9120537

Wang X, Zhang D, Su X, Li W (2017) Mlifdect: android malware detection based on parallel machine learning and information fusion. Security and Communication Networks 2017

Wolpert DH (1992) Stacked generalization. Neural networks 5(2):241–259

Wu D, Mao C, Wei T, Lee H, Wu K (2012) Droidmat: Android malware detection through manifest and api calls tracing. In: 2012 Seventh Asia Joint Conference on Information Security, pp 62–69, DOI 10.1109/AsiaJCIS.2012.18

Wu Y, Li X, Zou D, Yang W, Zhang X, Jin H (2019) Malscan: Fast market-wide mobile malware scanning by social-network centrality analysis. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp 139–150

Xu J, Li Y, Deng RH (2021) Differential training: A generic framework to reduce label noises for android malware detection. In: Proc. of Network and Distributed System Security Symposium (NDSS)

Yerima SY, Sezer S, Muttik I (2014) Android malware detection using parallel machine learning classifiers. In: 2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies, pp 37–42, DOI 10.1109/NGMAST.2014.23

Yerima SY, Sezer S, Muttik I (2015) High accuracy android malware detection using ensemble learning. IET Information Security 9(6):313–320

Zhang W, Ren H, Jiang Q, Zhang K (2015) Exploring feature extraction and elm in malware detection for android devices. In: Hu X, Xia Y, Zhang Y, Zhao D (eds) Advances in Neural Networks – ISNN 2015, Springer International Publishing, Cham, pp 489–498

Zhang X, Jin Z (2016) A new semantics-based android malware detection. In: 2016 2nd IEEE International Conference on Computer and Communications (ICCC), pp 1412–1416, DOI 10.1109/CompComm.2016.7924936

Zhao C, Zheng W, Gong L, Zhang M, Wang C (2018) Quick and accurate android malware detection based on sensitive apis. In: 2018 IEEE International Conference on Smart Internet of Things (SmartIoT), pp 143–148, DOI 10.1109/SmartIoT.2018.00034

Zhao C, Wang C, Zheng W (2019) Android malware detection based on sensitive permissions and apis. In: International Conference on Security and Privacy in New Computing Environments, Springer, pp 105–113

Zhao Y, Li L, Wang H, Cai H, Bissyandé TF, Klein J, Grundy J (2021) On the impact of sample duplication in machine-learning-based android malware detection. ACM Trans Softw

Eng Methodol 30(3), DOI 10.1145/3446905, URL https://doi.org/10.1145/3446905
Zhu H, Li Y, Li R, Li J, You Z, Song H (2020) Sedmdroid: An enhanced stacking ensemble of deep learning framework for android malware detection. IEEE Transactions on Network Science and Engineering pp 1–1, DOI 10.1109/TNSE.2020.2996379

10 Appendix

Table 7: Evaluation of the state-of-the-art approaches versus the combination of features versus the combination of classifiers on the whole `Literature` and `AndroZoo datasets`

		Literature whole dataset								**AndroZoo whole dataset**							
		Temporally-inconsistent				**Temporally-consistent**				**Temporally-inconsistent**				**Temporally-consistent**			
		R	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**
RQ1	**DREBIN**	0.91	0.93	**0.92**	0.96	0.64	0.34	**0.44**	0.64	0.95	0.98	0.96	0.98	0.76	0.98	0.85	0.92
	Reveal	0.55	0.87	0.68	0.88	0.51	0.3	0.38	0.63	0.92	0.99	0.95	0.97	0.82	0.98	0.89	0.94
	MaMaF	0.33	0.9	0.48	0.84	0.16	0.23	0.19	0.7	0.91	0.99	0.95	0.97	0.86	1	0.92	0.96
	MaMaP	0.56	0.97	0.71	0.9	0.28	0.18	0.22	0.57	0.96	0.97	0.96	0.98	0.89	0.96	0.92	0.96
	MalD	0.89	0.87	0.88	0.95	0.51	0.25	0.33	0.55	0.96	0.97	0.96	0.98	0.9	0.97	**0.93**	0.96
	MalH	0.89	0.88	0.89	0.95	0.42	0.27	0.33	0.62	0.96	0.97	0.97	0.98	0.9	0.96	0.93	0.96
	MalK	0.9	0.88	0.89	0.95	0.54	0.27	0.36	0.58	0.95	0.97	0.96	0.98	0.86	0.96	0.9	0.95
	MalCl	0.9	0.88	0.89	0.95	0.56	0.31	0.4	0.62	0.97	0.97	**0.97**	0.98	0.9	0.96	0.93	0.96
	MalA	0.9	0.89	0.89	0.95	0.44	0.28	0.34	0.63	0.95	0.96	0.96	0.97	0.87	0.96	0.91	0.95
	MalCo	0.9	0.89	0.89	0.95	0.44	0.28	0.34	0.63	0.96	0.97	0.96	0.98	0.86	0.96	0.91	0.95
RQ3	**LinearSVC**	0.87	0.72	0.78	0.88	0.7	0.38	0.48	0.65	0.92	0.82	0.87	0.92	0.83	0.8	0.81	0.89
	RF	0.87	0.96	0.91	0.96	0.49	0.31	0.38	0.64	0.97	0.99	0.98	0.99	0.88	0.99	0.93	0.96
	KNN	0.84	0.88	0.86	0.94	0.43	0.23	0.3	0.56	0.94	0.94	0.94	0.96	0.81	0.9	0.85	0.92
	AdaBoost	0.83	0.89	0.86	0.94	0.52	0.39	0.45	0.72	0.93	0.98	0.96	0.97	0.88	0.99	0.93	0.96
	Bagging	0.93	0.95	**0.94**	0.97	0.58	0.42	**0.49**	0.73	0.97	0.99	**0.98**	0.99	0.88	1	**0.94**	0.96
	GradBoosting	0.86	0.94	0.9	0.96	0.57	0.42	0.48	0.73	0.94	0.99	0.97	0.98	0.87	1	0.93	0.96
RQ4	**MajorVote**	0.89	0.94	0.92	0.96	0.39	0.29	0.33	0.66	0.96	0.99	0.97	0.98	0.87	1	0.93	0.96
	AvgProba	0.9	0.92	0.91	0.96	0.4	0.27	0.32	0.63	0.97	0.98	0.97	0.98	0.9	0.99	0.94	0.97
	AccWProba	0.9	0.92	0.91	0.96	0.4	0.27	0.32	0.63	0.97	0.98	0.97	0.98	0.9	0.99	0.94	0.97
	F1WProba	0.9	0.92	0.91	0.96	0.42	0.27	0.33	0.61	0.97	0.98	0.97	0.98	0.9	0.99	0.94	0.97
	MinProba	0.18	0.99	0.3	0.82	0.07	0.63	0.12	0.78	0.87	1	0.93	0.96	0.7	1	0.83	0.91
	MaxProba	0.98	0.72	0.83	0.91	0.93	0.26	0.4	0.39	0.99	0.9	0.94	0.96	0.92	0.85	0.89	0.93
	ProdProba	0.0	0.0	0.0	0.78	0.0	0.75	0.0	0.78	0.0	0.6	0.0	0.71	0.0	0.7	0.0	0.71
	StaPredSVM	0.93	0.92	0.93	0.97	0.56	0.24	0.33	0.51	0.95	0.98	0.96	0.98	0.76	0.98	0.85	0.92
	StaProbSVM	0.94	0.95	**0.94**	0.97	0.68	0.28	0.39	0.54	0.97	0.98	0.97	0.99	0.82	0.98	0.89	0.94
	StaPredRF	0.92	0.93	0.92	0.97	0.57	0.25	0.34	0.52	0.95	0.98	0.97	0.98	0.76	0.98	0.85	0.92
	StaProbRF	0.94	0.93	0.94	0.97	0.72	0.26	0.38	0.49	0.96	0.98	0.97	0.98	0.83	0.99	0.9	0.95
	StaPredKNN	0.92	0.93	0.92	0.97	0.65	0.28	0.39	0.55	0.96	0.98	0.97	0.98	0.76	0.98	0.86	0.92
	StaProbKNN	0.92	0.92	0.92	0.97	0.71	0.3	0.42	0.56	0.96	0.98	0.97	0.98	0.83	0.98	0.9	0.94
	StaPredMLP	0.93	0.93	0.93	0.97	0.65	0.27	0.38	0.53	0.96	0.98	0.97	0.98	0.76	0.98	0.86	0.92
	StaProbMLP	0.94	0.94	0.94	0.97	0.72	0.27	0.39	0.5	0.96	0.98	0.97	0.98	0.78	0.98	0.87	0.93
	EnsemSelect	0.92	0.96	0.94	0.97	0.48	0.39	**0.43**	0.71	0.97	0.99	**0.98**	0.99	0.89	1	**0.94**	0.97

The cells highlighted in grey show the best detector for each dataset and for each RQ based on the F1 score before rounding

Table 8: Evaluation of the state-of-the-art approaches versus the combination of features versus the combination of classifiers on the 2019 and 2020 `AndroZoo datasets`

		AndroZoo 2019 dataset								**AndroZoo 2020 dataset**							
		Temporally-inconsistent				**Temporally-consistent**				**Temporally-inconsistent**				**Temporally-consistent**			
		R	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**
RQ1	**DREBIN**	0.95	0.98	0.96	0.98	0.96	0.92	0.94	0.96	0.97	0.97	0.97	0.98	0.71	0.98	0.82	0.93
	Reveal	0.92	0.99	0.95	0.97	0.95	0.93	0.94	0.96	0.92	0.99	0.95	0.98	0.74	0.99	0.85	0.94
	MaMaF	0.9	0.99	0.95	0.97	0.97	0.99	**0.98**	0.99	0.94	1	0.97	0.99	0.76	1	0.86	0.95
	MaMaP	0.96	0.98	**0.97**	0.98	0.98	0.98	0.98	0.99	0.97	0.99	**0.98**	0.99	0.77	1	**0.87**	0.95
	MalD	0.96	0.97	0.96	0.98	0.98	0.94	0.96	0.97	0.96	0.97	0.97	0.99	0.78	0.99	0.87	0.95
	MalH	0.96	0.96	0.96	0.97	0.98	0.95	0.96	0.98	0.97	0.97	0.97	0.99	0.78	0.98	0.87	0.95
	MalK	0.95	0.97	0.96	0.98	0.97	0.93	0.95	0.97	0.96	0.97	0.97	0.99	0.78	0.97	0.86	0.94
	MalCl	0.96	0.97	0.96	0.98	0.98	0.95	0.96	0.97	0.97	0.97	0.97	0.99	0.78	0.98	0.87	0.95
	MalA	0.96	0.95	0.95	0.97	0.97	0.94	0.96	0.97	0.96	0.97	0.97	0.98	0.78	0.98	0.87	0.95
	MalCo	0.96	0.96	0.96	0.97	0.97	0.95	0.96	0.97	0.97	0.97	0.97	0.99	0.78	0.98	0.87	0.95
RQ3	**LinearSVC**	0.92	0.79	0.85	0.89	0.94	0.8	0.86	0.9	0.91	0.85	0.87	0.94	0.68	0.86	0.76	0.9
	RF	0.96	0.99	0.98	0.98	0.98	0.97	0.98	0.98	0.97	0.99	0.98	0.99	0.77	1	0.87	0.95
	KNN	0.94	0.94	0.94	0.96	0.94	0.83	0.88	0.92	0.93	0.92	0.93	0.97	0.63	0.88	0.74	0.9
	AdaBoost	0.93	0.97	0.95	0.97	0.98	0.96	0.97	0.98	0.95	0.99	0.97	0.99	0.77	0.99	0.86	0.95
	Bagging	0.97	0.99	**0.98**	0.99	0.98	0.98	0.98	0.99	0.98	0.99	**0.99**	0.99	0.77	0.99	**0.87**	0.95
	GradBoosting	0.94	0.99	0.97	0.98	0.98	0.98	**0.98**	0.99	0.96	0.99	0.98	0.99	0.77	1	0.87	0.95
RQ4	**MajorVote**	0.95	0.99	0.97	0.98	0.97	0.97	0.97	0.98	0.97	0.98	0.98	0.99	0.78	0.99	0.87	0.95
	AvgProba	0.96	0.97	0.97	0.98	0.98	0.96	0.97	0.98	0.97	0.98	0.97	0.99	0.78	0.99	0.87	0.95
	AccWProba	0.96	0.97	0.97	0.98	0.98	0.96	0.97	0.98	0.97	0.98	0.97	0.99	0.78	0.99	0.87	0.95
	F1WProba	0.96	0.97	0.97	0.98	0.98	0.96	0.97	0.98	0.97	0.98	0.97	0.99	0.78	0.99	0.87	0.95
	MinProba	0.88	1	0.93	0.96	0.92	1	0.96	0.97	0.88	1	0.93	0.97	0.65	1	0.79	0.92
	MaxProba	0.99	0.9	0.94	0.96	1	0.81	0.89	0.92	0.99	0.92	0.95	0.98	0.8	0.93	0.86	0.94
	ProdProba	0.0	0.6	0.0	0.67	0.0	0.0	0.0	0.67	0.0	0.1	0.0	0.77	0.0	0.0	0.0	0.78
	StaPredSVM	0.95	0.98	0.96	0.98	0.96	0.91	0.94	0.96	0.97	0.97	0.97	0.99	0.71	0.98	0.83	0.93
	StaProbSVM	0.96	0.98	0.97	0.98	0.97	0.92	0.95	0.96	0.98	0.99	0.98	0.99	0.74	0.99	0.85	0.94
	StaPredRF	0.95	0.98	0.97	0.98	0.96	0.91	0.94	0.96	0.97	0.97	0.97	0.99	0.72	0.98	0.83	0.93
	StaProbRF	0.96	0.98	0.97	0.98	0.97	0.93	0.95	0.97	0.98	0.98	0.98	0.99	0.75	0.99	0.85	0.94
	StaPredKNN	0.95	0.98	0.97	0.98	0.96	0.92	0.94	0.96	0.97	0.97	0.97	0.99	0.71	0.98	0.82	0.93
	StaProbKNN	0.96	0.98	0.97	0.98	0.97	0.92	0.94	0.96	0.97	0.98	0.98	0.99	0.78	0.98	0.87	0.95
	StaPredMLP	0.95	0.98	0.97	0.98	0.96	0.91	0.94	0.96	0.97	0.97	0.97	0.99	0.71	0.98	0.83	0.93
	StaProbMLP	0.96	0.98	0.97	0.98	0.97	0.92	0.95	0.96	0.97	0.98	0.98	0.99	0.75	0.99	0.85	0.94
	EnsemSelect	0.97	0.99	**0.98**	0.99	0.99	0.97	**0.98**	0.99	0.97	0.99	**0.98**	0.99	0.78	1	**0.88**	0.95

The cells highlighted in grey show the best detector for each dataset and for each RQ based on the F1 score before rounding

Table 9: Evaluation of the state-of-the-art approaches versus the combination of features versus the combination of classifiers on the DREBIN and RevealDroid `Literature datasets`

		Literature DREBIN dataset								**Literature RevealDroid dataset**							
		Temporally-inconsistent				**Temporally-consistent**				**Temporally-inconsistent**				**Temporally-consistent**			
		R	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**
RQ1	**DREBIN**	0.93	0.96	**0.94**	0.99	0.92	0.82	**0.87**	0.99	0.98	0.97	**0.97**	0.98	0.9	0.99	**0.94**	0.95
	Reveal	0.29	0.87	0.44	0.97	0.27	0.79	0.41	0.96	0.94	0.86	0.9	0.9	0.69	0.97	0.81	0.85
	MaMaF	0.18	1	0.31	0.96	0.19	1	0.32	0.96	0.95	0.85	0.9	0.9	0.83	0.93	0.88	0.89
	MaMaP	0.31	1	0.48	0.97	0.08	0.9	0.14	0.96	0.99	0.89	0.94	0.94	0.91	0.97	0.94	0.95
	MalD	0.89	0.85	0.87	0.99	0.87	0.58	0.7	0.97	0.95	0.93	0.94	0.94	0.76	0.98	0.86	0.89
	MalH	0.89	0.88	0.89	0.99	0.95	0.56	0.7	0.96	0.96	0.94	0.95	0.95	0.82	0.98	0.89	0.91
	MalK	0.91	0.89	0.9	0.99	0.88	0.57	0.69	0.96	0.96	0.94	0.95	0.95	0.81	0.98	0.89	0.91
	MalCl	0.9	0.86	0.88	0.99	0.87	0.69	0.77	0.98	0.96	0.94	0.95	0.95	0.81	0.99	0.89	0.91
	MalA	0.89	0.88	0.89	0.99	0.95	0.56	0.71	0.96	0.96	0.94	0.95	0.95	0.82	0.98	0.89	0.91
	MalCo	0.89	0.88	0.89	0.99	0.95	0.56	0.7	0.96	0.96	0.94	0.95	0.95	0.82	0.98	0.89	0.91
RQ3	**LinearSVC**	0.87	0.8	0.83	0.98	0.97	0.6	0.73	0.97	0.95	0.93	0.94	0.94	0.8	0.96	0.87	0.89
	RF	0.85	1	0.92	0.99	0.75	1	0.85	0.99	0.96	0.97	0.97	0.97	0.66	1	0.79	0.84
	KNN	0.8	0.88	0.83	0.99	0.79	0.73	0.76	0.98	0.94	0.91	0.92	0.93	0.57	0.97	0.72	0.8
	AdaBoost	0.81	0.92	0.86	0.99	0.88	0.94	**0.91**	0.99	0.97	0.98	0.97	0.98	0.92	0.99	0.95	0.96
	Bagging	0.94	0.97	**0.96**	1	0.77	0.95	0.85	0.99	0.98	0.98	**0.98**	0.98	0.89	1	0.94	0.95
	GradBoosting	0.86	0.98	0.92	0.99	0.71	0.98	0.82	0.99	0.98	0.98	0.98	0.98	0.93	1	**0.96**	0.96
RQ4	**MajorVote**	0.88	0.97	0.92	0.99	0.86	0.84	0.85	0.99	0.97	0.96	0.96	0.97	0.84	0.99	0.91	0.93
	AvgProba	0.89	0.95	0.92	0.99	0.87	0.8	0.83	0.98	0.96	0.95	0.96	0.96	0.84	0.99	0.91	0.92
	AccWProba	0.89	0.95	0.92	0.99	0.87	0.81	0.84	0.98	0.96	0.95	0.96	0.96	0.84	0.99	0.91	0.92
	F1WProba	0.89	0.94	0.91	0.99	0.87	0.76	0.81	0.98	0.96	0.95	0.96	0.96	0.84	0.99	0.91	0.92
	MinProba	0.08	1	0.14	0.96	0.0	0.1	0.0	0.95	0.85	1	0.92	0.93	0.46	1	0.63	0.75
	MaxProba	0.97	0.69	0.81	0.98	0.99	0.38	0.54	0.92	1	0.76	0.86	0.86	0.98	0.91	0.95	0.95
	ProdProba	0.0	0.0	0.0	0.95	0.0	0.0	0.0	0.95	0.0	0.3	0.0	0.54	0.0	0.0	0.0	0.54
	StaPredSVM	0.94	0.93	0.93	0.99	0.95	0.72	0.82	0.98	0.99	0.96	0.97	0.98	0.92	0.99	0.95	0.96
	StaProbSVM	0.94	0.97	**0.95**	1	0.9	0.79	0.84	0.98	0.98	0.97	0.97	0.97	0.86	0.99	0.92	0.94
	StaPredRF	0.94	0.91	0.93	0.99	0.95	0.67	0.79	0.98	0.99	0.96	0.97	0.97	0.89	0.99	0.94	0.95
	StaProbRF	0.95	0.95	0.95	1	0.85	0.76	0.8	0.98	0.99	0.96	0.97	0.98	0.91	0.99	0.94	0.95
	StaPredKNN	0.94	0.91	0.92	0.99	0.93	0.68	0.79	0.98	0.97	0.97	0.97	0.97	0.85	0.99	0.92	0.93
	StaProbKNN	0.95	0.93	0.94	0.99	0.88	0.73	0.8	0.98	0.97	0.95	0.96	0.96	0.86	0.99	0.92	0.93
	StaPredMLP	0.95	0.91	0.93	0.99	0.97	0.68	0.8	0.98	0.98	0.97	0.97	0.98	0.89	0.99	0.94	0.95
	StaProbMLP	0.95	0.95	0.95	1	0.91	0.72	0.81	0.98	0.99	0.95	0.97	0.97	0.91	0.99	0.95	0.96
	EnsemSelect	0.93	0.97	0.95	1	0.96	0.84	**0.89**	0.99	0.99	0.98	**0.98**	0.98	0.94	0.99	**0.96**	0.97

The cells highlighted in grey show the best detector for each dataset and for each RQ based on the F1 score before rounding

Table 10: Evaluation of the state-of-the-art approaches versus the combination of features versus the combination of classifiers on the MaMaDroid and MalScan `Literature datasets`

		Literature MaMaDroid dataset								**Literature MalScan dataset**							
		Temporally-inconsistent				**Temporally-consistent**				**Temporally-inconsistent**				**Temporally-consistent**			
		R	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**	**R**	**P**	**F1**	**A**
RQ1	**DREBIN**	0.98	0.98	**0.98**	0.96	0.89	0.95	**0.92**	0.87	0.96	0.96	**0.96**	0.96	0.78	0.95	0.86	0.87
	Reveal	0.97	0.92	0.94	0.91	0.73	0.94	0.83	0.75	0.87	0.91	0.89	0.9	0.48	0.88	0.62	0.72
	MaMaF	0.98	0.9	0.94	0.89	0.86	0.92	0.89	0.82	0.81	0.84	0.82	0.83	0.53	0.82	0.64	0.72
	MaMaP	0.98	0.92	0.95	0.92	0.78	0.93	0.85	0.78	0.94	0.96	0.95	0.95	0.61	0.98	0.74	0.81
	MalD	0.96	0.95	0.95	0.93	0.68	0.94	0.79	0.71	0.97	0.93	0.95	0.95	0.81	0.95	0.87	0.89
	MalH	0.96	0.96	0.96	0.93	0.75	0.95	0.83	0.76	0.97	0.95	0.96	0.96	0.82	0.96	**0.88**	0.9
	MalK	0.96	0.95	0.96	0.93	0.71	0.95	0.81	0.73	0.97	0.94	0.95	0.96	0.8	0.95	0.87	0.88
	MalCl	0.96	0.95	0.96	0.93	0.75	0.94	0.84	0.77	0.97	0.94	0.96	0.96	0.82	0.96	0.88	0.89
	MalA	0.96	0.96	0.96	0.94	0.75	0.95	0.83	0.76	0.97	0.95	0.96	0.96	0.8	0.96	0.87	0.89
	MalCo	0.96	0.96	0.96	0.94	0.74	0.95	0.83	0.76	0.97	0.95	0.96	0.96	0.8	0.96	0.87	0.89
RQ3	**LinearSVC**	0.92	0.94	0.93	0.89	0.72	0.92	0.81	0.72	0.88	0.9	0.89	0.89	0.65	0.82	0.71	0.76
	RF	0.97	0.96	0.97	0.95	0.64	0.95	0.77	0.69	0.96	0.98	0.97	0.97	0.76	0.99	0.86	0.88
	KNN	0.96	0.93	0.95	0.92	0.62	0.93	0.74	0.65	0.93	0.89	0.91	0.91	0.75	0.9	0.82	0.84
	AdaBoost	0.96	0.96	0.96	0.94	0.85	0.95	0.9	0.84	0.95	0.95	0.95	0.95	0.74	0.91	0.81	0.84
	Bagging	0.97	0.97	**0.97**	0.96	0.78	0.95	0.86	0.79	0.98	0.97	**0.97**	0.98	0.78	0.94	0.85	0.87
	GradBoosting	0.97	0.96	0.97	0.95	0.85	0.95	**0.9**	0.85	0.97	0.97	0.97	0.97	0.81	0.95	**0.88**	0.89
RQ4	**MajorVote**	0.97	0.96	0.97	0.95	0.76	0.95	0.85	0.78	0.97	0.97	0.97	0.97	0.78	0.99	0.87	0.89
	AvgProba	0.97	0.96	0.96	0.94	0.76	0.95	0.85	0.78	0.97	0.96	0.97	0.97	0.81	0.98	0.89	0.9
	AccWProba	0.97	0.96	0.96	0.94	0.76	0.95	0.85	0.78	0.97	0.96	0.97	0.97	0.81	0.98	0.89	0.9
	F1WProba	0.97	0.96	0.96	0.94	0.76	0.95	0.85	0.78	0.97	0.96	0.97	0.97	0.81	0.98	0.89	0.9
	MinProba	0.9	0.99	0.94	0.91	0.45	0.97	0.62	0.55	0.69	1	0.82	0.85	0.25	1	0.38	0.64
	MaxProba	1	0.87	0.93	0.88	0.96	0.91	**0.93**	0.89	1	0.76	0.86	0.85	0.94	0.78	0.85	0.84
	ProdProba	0.0	0.5	0.0	0.2	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.52	0.0	0.0	0.0	0.52
	StaPredSVM	0.98	0.97	0.97	0.96	0.81	0.95	0.87	0.81	0.96	0.96	0.96	0.96	0.78	0.93	0.85	0.87
	StaProbSVM	0.98	0.97	0.97	0.96	0.75	0.95	0.84	0.77	0.98	0.98	0.98	0.98	0.84	0.98	**0.9**	0.91
	StaPredRF	0.98	0.96	0.97	0.95	0.82	0.95	0.88	0.82	0.96	0.98	0.97	0.97	0.76	0.96	0.85	0.87
	StaProbRF	0.98	0.97	0.97	0.96	0.84	0.95	0.89	0.84	0.98	0.97	0.97	0.98	0.81	0.98	0.89	0.9
	StaPredKNN	0.98	0.97	0.97	0.95	0.8	0.95	0.87	0.81	0.95	0.98	0.97	0.97	0.76	0.99	0.86	0.88
	StaProbKNN	0.97	0.97	0.97	0.95	0.74	0.95	0.83	0.76	0.97	0.97	0.97	0.97	0.83	0.97	0.9	0.91
	StaPredMLP	0.98	0.97	0.97	0.95	0.83	0.95	0.88	0.83	0.96	0.98	0.97	0.97	0.76	0.97	0.85	0.87
	StaProbMLP	0.98	0.97	0.97	0.95	0.77	0.96	0.85	0.79	0.98	0.97	0.97	0.97	0.82	0.97	0.89	0.9
	EnsemSelect	0.98	0.97	**0.98**	0.96	0.89	0.94	0.91	0.86	0.98	0.98	**0.98**	0.98	0.82	0.98	0.89	0.9

The cells highlighted in grey show the best detector for each dataset and for each RQ based on the F1 score before rounding

Table 11: Proportion of malware samples detected by state-of-the-art approaches and belonging to the 20 top families in the `Literature dataset`

	Families	#	DREBIN	Reveal	MaMaF	MaMaP	MalD	MalH	MalK	MalCl	MalA	MalCo
Temporally inconsistent	dowgin	352	**98.6**	68.8	63.1	86.9	93.2	94.0	94.3	94.0	94.0	94.0
	fakeinst	345	**99.4**	31.9	28.7	38.0	98.6	98.6	98.6	98.6	98.6	98.6
	kuguo	246	**96.7**	62.2	62.6	83.3	94.7	94.7	93.9	95.1	94.7	94.7
	smsreg	222	**96.4**	79.3	43.7	67.6	91.4	93.2	92.8	93.2	93.2	92.8
	airpush	214	86.4	57.0	6.1	72.4	87.4	**91.6**	90.2	91.1	**91.6**	90.7
	gappusin	199	86.4	66.8	28.6	45.2	88.9	89.4	**89.9**	88.9	89.4	89.4
	boxer	195	99.5	1.5	76.9	82.1	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	adwo	195	82.6	72.8	11.8	62.6	85.6	86.2	**86.7**	85.6	86.2	86.2
	opfake	156	**100.0**	69.2	2.6	8.3	98.7	82.7	98.7	98.1	98.1	98.7
	plankton	119	**98.3**	28.6	3.4	58.8	90.8	90.8	94.1	92.4	90.8	90.8
	youmi	111	**82.9**	66.7	38.7	51.4	82.0	**82.9**	85.6	**82.9**	**82.9**	**82.9**
	umpay	89	**98.9**	92.1	49.4	94.4	97.8	97.8	**98.9**	**98.9**	97.8	97.8
	droidkungfu	80	92.5	73.8	52.5	66.2	92.5	92.5	**93.8**	**93.8**	**93.8**	**93.8**
	secapk	66	**95.5**	93.9	6.1	13.6	75.8	75.8	74.2	75.8	75.8	74.2
	domob	54	85.2	77.8	22.2	25.9	85.2	85.2	**87.0**	**87.0**	85.2	85.2
	smsagent	54	**98.1**	5.6	0.0	1.9	96.3	94.4	96.3	94.4	94.4	94.4
	admogo	50	**100.0**	96.0	24.0	90.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	revmob	48	81.2	45.8	50.0	83.3	81.2	**85.4**	**85.4**	**85.4**	81.2	83.3
	ginmaster	48	**83.3**	14.6	14.6	25.0	77.1	81.2	79.2	81.2	81.2	81.2
	jiagu	45	97.8	93.3	13.3	0.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Temporally consistent	jiagu	408	**77.2**	64.0	0.0	0.2	0.7	10.3	0.7	0.7	19.9	19.9
	dnotua	303	5.0	5.0	0.3	11.2	**94.4**	2.6	94.1	94.1	2.6	2.6
	airpush	296	60.5	38.5	3.0	60.8	69.9	75.7	**87.2**	68.2	75.7	75.7
	fakeapp	222	8.1	6.3	1.8	1.4	**59.9**	24.8	32.4	59.0	24.8	24.8
	smsreg	136	**94.1**	77.9	36.0	52.2	57.4	66.9	63.2	63.2	69.1	69.1
	secapk	122	40.2	**92.6**	4.9	4.9	41.8	43.4	43.4	43.4	43.4	43.4
	smspay	107	**96.3**	88.8	38.3	56.1	78.5	80.4	79.4	78.5	81.3	81.3
	hiddenads	80	**98.8**	63.8	7.5	7.5	15.0	23.8	16.2	17.5	30.0	30.0
	dowgin	74	**86.5**	**86.5**	45.9	51.4	54.1	63.5	64.9	67.6	64.9	64.9
	kyview	70	**98.6**	71.4	31.4	41.4	52.9	58.6	51.4	54.3	62.9	62.9
	tencentprotect	69	68.1	88.4	0	0	97.1	**98.6**	**98.6**	**98.6**	**98.6**	**98.6**
	youmi	63	**98.4**	66.7	50.8	55.6	47.6	61.9	49.2	55.6	65.1	65.1
	ramnit	59	28.8	13.6	8.5	25.4	71.2	16.9	35.6	**76.3**	16.9	16.9
	kuguo	50	**88.0**	78.0	72.0	66.0	58.0	48.0	52.0	58.0	48.0	48.0
	ewind	47	**59.6**	**59.6**	6.4	10.6	46.8	51.1	34.0	36.2	55.3	55.3
	leadbolt	45	**88.9**	84.4	4.4	71.1	80.0	73.3	77.8	75.6	73.3	73.3
	revmob	45	95.6	17.8	0.0	35.6	**100.0**	88.9	82.2	95.6	88.9	88.9
	tachi	41	0	0	0	0	0	0	0	0	0	0
	feiwo	35	54.3	51.4	48.6	**65.7**	60.0	51.4	57.1	60.0	51.4	51.4
	umpay	32	93.8	93.8	56.2	75.0	68.8	78.1	**96.9**	78.1	78.1	78.1

The entries in bold show the best detector for each malware family

Table 12: Proportion of malware samples detected by state-of-the-art approaches and belonging to the 20 top families in the `AndroZoo dataset`

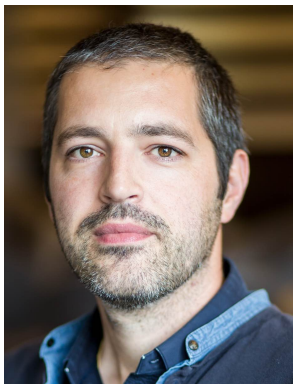
	Families	#	DREBIN	Reveal	MaMaF	MaMaP	MalD	MalH	MalK	MalCl	MalA	MalCo
Temporally inconsistent	jiagu	4781	99.6	99.9	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	dnotua	140	43.6	0.7	92.1	**97.9**	97.1	96.4	48.6	96.4	52.1	52.9
	secneo	69	84.1	**98.6**	0	**98.6**	79.7	58.0	79.7	79.7	78.3	78.3
	tencentprotect	57	**100.0**	96.5	**100.0**	**100.0**	98.2	98.2	98.2	98.2	98.2	98.2
	smsreg	46	**95.7**	78.3	19.6	82.6	**95.7**	**95.7**	**95.7**	**95.7**	**95.7**	**95.7**
	hiddad	16	**81.2**	37.5	6.2	56.2	75.0	75.0	75.0	68.8	75.0	75.0
	smspay	11	**100.0**	**100.0**	90.9	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	hypay	9	**100.0**	77.8	0	88.9	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	utilcode	9	**100.0**	**100.0**	0.0	88.9	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	ewind	8	**100.0**	62.5	75.0	**100.0**	75.0	75.0	87.5	75.0	75.0	75.0
	wapron	8	**100.0**	**100.0**	87.5	87.5	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	datacollector	7	**100.0**	85.7	0.0	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	kuguo	6	**83.3**	**83.3**	0.0	66.7	66.7	66.7	66.7	66.7	66.7	66.7
	styricka	6	**100.0**	83.3	0	0.0	83.3	83.3	83.3	83.3	83.3	83.3
	fakeapp	5	80.0	0.0	0.0	80.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	triada	5	**100.0**	80.0	60.0	80.0	80.0	80.0	80.0	80.0	80.0	80.0
	revmob	5	**80.0**	**80.0**	0	**80.0**	**80.0**	**80.0**	**80.0**	**80.0**	**80.0**	**80.0**
	airpush	5	**60.0**	20.0	0	40.0	**60.0**	**60.0**	**60.0**	**60.0**	**60.0**	**60.0**
	baiduprotect	4	75.0	75.0	0.0	**100.0**	0.0	25.0	25.0	25.0	25.0	25.0
	autoins	4	**100.0**	**100.0**	25.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
Temporally consistent	jiagu	5351	87.1	98.9	99.9	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	dnotua	457	28.9	0	71.8	**89.7**	88.8	89.1	29.8	87.7	35.9	34.6
	hiddenad	35	65.7	57.1	54.3	57.1	65.7	65.7	65.7	**68.6**	65.7	65.7
	hiddad	32	0.0	3.1	0	3.1	15.6	**21.9**	15.6	**21.9**	**21.9**	**21.9**
	joker	11	27.3	0	0	0	63.6	63.6	63.6	**72.7**	63.6	63.6
	jocker	9	0.0	0.0	0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	emagsoftware	9	**66.7**	33.3	0	0	11.1	11.1	11.1	11.1	11.1	11.1
	autoins	7	**100.0**	**100.0**	0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	smspay	7	**85.7**	**85.7**	71.4	**85.7**	71.4	71.4	71.4	71.4	71.4	71.4
	plankton	7	0	0	0	0	0	0	0	0	0	0
	hiddenads	6	0.0	0.0	0	0.0	**16.7**	0.0	0.0	0.0	0.0	0.0
	utilcode	6	**100.0**	33.3	0	16.7	66.7	50.0	66.7	50.0	50.0	50.0
	funpay	5	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	secneo	5	0.0	0	0	0	0	**20.0**	0	0	0	0
	fakeapp	5	0	0	0	0	0	0.0	0	0	0.0	0.0
	smsreg	5	40.0	40.0	40.0	20.0	40.0	40.0	**60.0**	**60.0**	40.0	40.0
	datacollector	5	**100.0**	40.0	0	80.0	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
	hiddenapp	5	0	0	0	0	0	0	0	0	0	0
	apkprotector	4	25.0	0.0	25.0	25.0	50.0	50.0	25.0	25.0	**50.0**	**50.0**
	airpush	4	**50.0**	25.0	0	0	25.0	25.0	25.0	25.0	25.0	25.0

The entries in bold show the best detector for each malware family

Nadia Daoudi received her Master degree in Computer Science from the École des Mines de Saint-Étienne (France), in 2018. She is currently a doctoral researcher at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) of the University of Luxembourg. Her research interests are in the area of Android security with a special focus on Machine Learning-based Android malware detection.

Kevin Allix is a Research Associate at the SnT - University of Luxembourg, where he carries research on Android Malware detection, Machine-Learning for Security, Software Engineering, and Natural Language Processing. Before he moved to research, Kevin held operational positions in network, system, and security engineering. Kevin received his PhD degree in 2015 from the University of Luxembourg.

Tegawendé F. Bissyandé is a chief scientist at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) of the University of Luxembourg. He received his PhD degree in Computer Sciences from the University of Bordeaux (France) in 2013. His research interests lie in trustworthy software engineering, notably in automated debugging, automated program repair and software security. He has co-authored over 80 research papers, in top-tiers venues such as TSE, TOSEM, ICSE, ESEC/FSE, ASE.



Jacques Klein is a chief scientist at the Interdisciplinary Centre for Security, Reliability and Trust (SnT) of the University of Luxembourg. He received a Ph.D. degree in Computer Science from the University of Rennes, France in 2006. His main areas of expertise are: Software Security, Software reliability, and explainable software. He has published over 150 research papers, often at prestigious venues such as TSE, ICSE, ESEC/FSE, Usenix Security, etc.