A Deep Dive Inside DREBIN: An Explorative Analysis beyond Android Malware Detection Scores

NADIA DAOUDI, KEVIN ALLIX, TEGAWENDÉ FRANÇOIS BISSYANDÉ, and JACQUES KLEIN, University of Luxembourg

Machine learning advances have been extensively explored for implementing large-scale malware detection. When reported in the literature, performance evaluation of machine learning based detectors generally focuses on highlighting the ratio of samples that are correctly or incorrectly classified, overlooking essential questions on why/how the learned models can be demonstrated as reliable. In the Android ecosystem, several recent studies have highlighted how evaluation setups can carry biases related to datasets or evaluation methodologies. Nevertheless, there is little work attempting to dissect the produced model to provide some understanding of its intrinsic characteristics. In this work, we fill this gap by performing a comprehensive analysis of a state-of-the-art Android malware detector, namely DREBIN, which constitutes today a key reference in the literature. Our study mainly targets an in-depth understanding of the classifier characteristics in terms of (1) which features actually matter among the hundreds of thousands that DREBIN extracts, (2) whether the high scores of the classifier are dependent on the dataset age, and (3) whether DREBIN's explanations are consistent within malware families, among others. Overall, our tentative analysis provides insights into the discriminatory power of the feature set used by DREBIN to detect malware. We expect our findings to bring about a systematisation of knowledge for the community.

CCS Concepts: • **Security and privacy → Malware and its mitigation**; • **Computing methodologies → Machine learning;**

Additional Key Words and Phrases: Android malware detection, machine learning, DREBIN, SVM

ACM Reference format:
Nadia Daoudi, Kevin Allix, Tegawendé François Bissyandé, and Jacques Klein. 2022. A Deep Dive Inside DREBIN: An Explorative Analysis beyond Android Malware Detection Scores. *ACM Trans. Priv. Secur.* 25, 2, Article 13 (May 2022), 28 pages.
https://doi.org/10.1145/3503463

1 INTRODUCTION

Machine learning (ML) has been widely proposed as a promising technique to address the proliferation of Android malware through rapid, systematic, and large-scale identification of

This work was partly supported by the Luxembourg National Research Fund (FNR), under the project CHARACTERIZE C17/IS/11693861, by the SPARTA project, which has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 830892, by the University of Luxembourg, under the internal project HitDroid, and by the Luxembourg Ministry of Foreign and European Aairs through their Digital4Development (D4D) portfolio under project LuxWAyS.
Authors' address: N. Daoudi, K. Allix, T. F. Bissyandé, and J. Klein, SnT, University of Luxembourg, 29, Avenue J.F Kennedy, Luxembourg, Luxembourg, L-1359; emails: {nadia.daoudi, kevin.allix, tegawende.bissyande, jacques.klein}@uni.lu.

2471-2566/2022/05-ART13 $15.00
https://doi.org/10.1145/3503463

malicious samples and their variants. Several approaches [7, 8, 16, 28, 30, 33, 52] have then been presented in the literature, where authors explore static, dynamic or hybrid analysis methods to extract features, as well as a variety of classification algorithms. Although the malicious characteristic of a sample is derived from the app behaviour, many approaches rely only on static analysis to produce features that are then assumed to be a proxy representation of the app's entire code (and hence its behaviour). Such features unfortunately capture only a small predefined portion of information about the app behaviour: features select only a finite set of code attributes, may leave out native code, and so forth. Therefore, extracted features for ML-based malware detection offer an imprecise representation of an incomplete view of the app that is further used as a proxy of application behaviour. Consequently, there is an assumption that this proxy representation still contains enough of the relevant information for discriminating malware. Feature engineering is thus the essential step in malware detection that implements the intelligence of the malware detection approach, beyond the off-the-shelf classification algorithms that are employed.

In the literature of malware detection, state-of-the-art approaches mainly differ by the feature sets that are proposed. For example, DREBIN [7] is a key reference in the literature that builds on a standard SVM algorithm but was novel in proposing a large and domain-specific feature set that was manually engineered for the problem of Android malware detection. Interestingly, the contributions that stand out in most of the ML-based malware detection approaches are associated to the feature set. These key contributions, however, are rarely evaluated through in-depth ablation studies, where the feature set value is assessed thoroughly. Instead, the literature provides assessment results by reporting detection performance measurements of the overall approach. The state of the practice thus considers that good performance indicators offer sufficient validation on the value of the feature set.

Evaluation of ML-based malware detection approaches have been scrutinised in recent studies. Allix et al. [3] have demonstrated that the performance of a malware detector may drop drastically when it is evaluated in the wild. Spatial and temporal biases have also been pointed out in two independent research works [4, 35]. Although these works attempt initial analyses of *whether* models can fail to generalise, they do not provide in-depth investigations into *how* or *why*. We propose to fill this gap in our work by conducting a tentative analysis of the feature set that is fed to the learners.

Analysing a classifier is an open problem [27]. Our work engages in building a roadmap in this direction for ML-based malware detection. We propose a dissection study that overviews the performance of a classifier from different points of views. Our work builds on a reproduction study [11] that has considered Android malware detectors from 16 major venues on Software engineering, security, and ML. Specifically, we focus on the state-of-the-art malware detector DREBIN, which is a the most cited approach that has been successfully replicated. Typically, our investigation attempts to go beyond the quantitative measurements of precision and recall detection metrics, to highlight other qualitative dimensions of the approach. In particular, we seek answers to the following questions:

- What are the key app features that guarantee to DREBIN its high-performing detection scores?
- Are all the features needed to achieve state-of-the-art detection performance?
- Does DREBIN learn the concept of malware family?

To support that the classification decisions are dependable, beyond the high performance scores, DREBIN outputs some explanations which show that the model has reasonably captured

information that is relevant to the accuracy of the decision. We propose to dissect these explanations further through various analyses:

- To what extent are explanations given by DREBIN indicative of the classification decision?
- How consistent are the DREBIN explanations across samples of the same malware family?

Our study explores DREBIN along with its initial dataset but also considered three other datasets that were carefully collected to assess the generalisability of the findings. Our insights with this study will serve the community for better understanding the strengths and limitation of the DREBIN contribution, and thus offer a systematisation of knowledge around DREBIN. We expect this study to better drive the exploitation of DREBIN as a key contribution in the literature towards opening novel research directions and producing reliable and effective models for malware detection.

Among other findings, the study yields that:

❶ The feature set of DREBIN is sufficiently generic to capture enough concepts that are relevant to a diverse set of malware samples across time.
❷ The feature set of DREBIN contains a huge number of id-features (e.g., features such as component names). However, the relevance and importance of each feature remains challenging to quantify w.r.t. the overall performance of DREBIN.
❸ Most features in DREBIN are at best redundant and at worst useless. Indeed, a subset of DREBIN features, smaller than the whole set by three orders of magnitude, is enough to provide similar performance with the whole feature set.
❹ DREBIN includes some features which, *singlely*, can offer a surprisingly high detection rates on some datasets.
❺ DREBIN explanations do not reflect how much the features contribute to the prediction.
❻ DREBIN explanations are often inconsistent across samples from a specific malware family.

1.1 Background on DREBIN

DREBIN is an Android malware detector that makes use of static analysis and ML techniques to decide if a given Android application is likely to be malware or goodware. This approach has been developed and validated using 5,560 malware samples and 123,453 goodware apps whose compilation dates are all within the period from August 2010 to October 2012.

Leveraging the Manifest file[1] which is included in each Android app package, DREBIN extracts four sets of string features from this file using the Android Asset Packaging Tool: Hardware components, Requested permissions, App components, and Filtered intents. DREBIN further considers the information contained in the disassembled code of the apps to extract four additional sets of string features: Restricted API calls, Used permissions, Suspicious API calls, and Network addresses.

To feed the extracted sets of features to the classifier, a multi-dimensional vector space has to be created, using the combination of all the features, from the eight categories, that are extracted from the training apps. DREBIN feeds the multi-dimensional vectors of the training apps to a Linear SVM classifier to make it learn the relationship between the inputs (features vectors) and the outputs (samples are either malware or goodware). The trained classifier is then used to predict the class of new and unseen Android apps (i.e., the test set). In DREBIN's paper, the authors state that this classifier detects malware apps with a recall of 0.94. The ML algorithm used by DREBIN, SVM, has already been used by many prior works on Android malware detection. Therefore, we assume

[1] https://developer.android.com/guide/topics/manifest/manifest-intro.

that the defining contribution that gave DREBIN its performance are attributable to DREBIN's selection of feature rather than to its choice of algorithm.

Besides its effectiveness in detecting malware apps, DREBIN has made a breakthrough in the field by providing explanations of its decisions. DREBIN relies on the weights of the Linear SVM classifier to determine the features that contributed the most to the prediction. The notion of explainability is extended to malware families, where DREBIN explains a family based on the explanations given to the malware samples of that family.

2 DATASET AND STATISTICAL ANALYSIS OF DREBIN FEATURES

2.1 Dataset

To collect our dataset, we have mainly relied on AndroZoo, which is a large collection of more than 16 million Android apps and regularly growing [5]. Our dataset is built by considering malware and goodware samples that span several years (2017, 2018, and 2019), allowing to ensure that the insights that we draw are related to the properties of DREBIN, not the properties of a given dataset.

AndroZoo makes available, for each app it stores, the number $vt\_detection$ that represents the number of VirusTotal[2]-hosted antivirus that have detected this app as malware. Our goodware samples are defined as the apps that have $vt\_detection = 0$, to ensure that they have not been flagged by any antivirus engine. Following on past studies, we have set the $vt\_detection$ value to 6, which means that our malware apps have been detected by at least six antivirus engines from VirusTotal.

At the time of the experiments, we were able to collect 15,892 malware apps for the 2019 dataset. The same number of malware apps (i.e., 15,892) was collected for the 2018 and 2017 datasets to have comparable settings. As for the goodware, we have considered all the apps that meet our criteria (i.e., the apps that have $vt\_detection = 0$). Consequently, we have collected 175,327, 297,272, and 276,750 apps for 2019, 2018, and 2017 datasets, respectively.

We name `2019_data`, `2018_data`, and `2017_data` the collection of malware and goodware samples that have been collected for 2019, 2018, and 2017 datasets, respectively.

In addition to the three recent datasets we collect (`2019_data`, `2018_data`, and `2017_data`), we also conduct our analysis on DREBIN's original dataset to verify that our results are generic and stable across the time. To this end, we have leveraged DREBIN's malware dataset, which consists of 5,560 malware apps provided by the original authors upon request. As for the goodware dataset, the raw apps are not provided directly by the authors. We have searched them in AndroZoo using the list of the APKs' SHA256 hashes provided by the authors. Consequently, we were able to collect 57,307 apps (i.e., only 46.42% of the total number of goodware samples used in DREBIN's paper). To have a dataset that is similar to the one used in the original paper, we have complemented the goodware samples with 66,146 apps from the same period (i.e., August 2010 to October 2012), and having $vt\_detection = 0$. This dataset is denoted as `DREBIN_Like_data`. We note that 81 out of 5,560 malware applications have failed in the features extraction process. Table 1 summarises the number of applications in our collected datasets.

2.2 DREBIN Replication

To conduct our experiments, we replicated the DREBIN approach [11] and checked its performance using our datasets. We aim to assess DREBIN's performance as described in the original publication to compare our analysis with the results of this implementation.

[2]VirusTotal (https://www.virustotal.com) is an online service that allows to collect antivirus reports on uploaded samples.

Table 1. Number of Collected Malware and Goodware Apps in Our Datasets

Dataset	Malware	Goodware
2019_data	15,892	175,327
2018_data	15,892	297,272
2017_data	15,892	276,750
DREBIN_Like_data	5,560	123,453

Table 2. Number of Features and Performance Scores of Our Replication of DREBIN on Each Dataset

	2019_data	2018_data	2017_data	DREBIN_Like_data
# Features	1,230,854	1,331,583	1,486,191	389,957
Recall	0.974	0.924	0.918	0.933
Precision	0.983	0.926	0.938	0.955
F1-score	0.979	0.925	0.928	0.944

Hypothesis 1

DREBIN effectiveness is not specifically tied to its original dataset.

We split each of our four datasets into subsets (80% for training and 20% for testing). We then fit Linear SVM classifiers on the training datasets and calculate the performance scores on the test subsets. We note that all our experiments are performed using the well-known ML framework Scikit-learn.[3] We report in Table 2 the number of features that are used to perform the classification for our four dataset, as well as the three performance measures: recall, precision, and F1-score.

As we can see from the table, the three performance measures for the four datasets are very high, especially for `2019_data`, which reports an F1-score of 0.979 on a test dataset of 58,529 Android apps. These scores confirm that DREBIN performs very well, not only with the `DREBIN_Like_data` but also when evaluated with our own collected samples. DREBIN was proposed in 2014 with a dataset of malware and goodware samples that belong to the period from August 2010 to October 2012, and here it still performs very well with our recent collected datasets.

Finding 1

DREBIN provides excellent scores even when it is used with recent datasets (i.e., after retraining). This suggests that the feature set is sufficiently generic to capture enough concepts that are relevant to a large variety of malware along their variations across time, as long as retraining is performed in the same time frame.

Evaluating the performance of malware classifiers—like all security systems—is not a straightforward task [44]. Some experiments may provide brilliant results, but still they do not reflect how good the classifier is when it is deployed in realistic settings. The results presented in Table 2 show the performance of DREBIN when the training and test datasets belong to the same year. However, when DREBIN is evaluated with training apps that are temporally precedent to testing apps, TESSERACT [35] has reported that the performance of this classifier drops remarkably. The

[3] https://scikit-learn.org.

realistic setting of using Android malware detectors showed the limitation of DREBIN, which questions the way it works and to what extent it captures the behaviour of Android apps. Inspired by the results of TESSERACT [35], we aim to perform an in-depth study on DREBIN to have a better understanding of its learning.

2.3 Statistical Analysis of the Features

Based on the results presented in Table 2, we can see that the four classifiers use a large number of features to perform the classification. If we take the example of `2017_data`, DREBIN uses 1,486,191 features, and hence for each sample it creates a 1,486,191-dimensional vector as part of its learning/predicting process. This huge number of features is explained by the fact that DREBIN extracts as much features as it finds in the apps, and it does not make any cleaning nor evaluation of the importance nor the relevance of these features. In practice, not every feature in a dataset carries information useful for discriminating samples; some features might be redundant or irrelevant, and they may add randomness to the results. Given that DREBIN trains an SVM classifier with more than 1 million features, it is unclear what exact features are responsible for making SVM decide how to classify a given Android app, nor how many features (out of 1 million) are actually needed to enable it to make this decision.

Finding 2

DREBIN relies on a huge number of features. The relevance and importance of each feature, however, remains challenging to quantify w.r.t. the overall performance of DREBIN.

Another issue that needs to be discussed is related to the *quality* of some features used by DREBIN. As we have presented earlier, DREBIN uses string features that belong to `App components` and `Network addresses` categories. For instance, the names of Android components[4] are extracted from the apps. Since these components names are defined and attributed by the developers of the app, using them as features increases the possibility of DREBIN to overfit the dataset and decreases its capability to generalise. Malware developers can easily change the names of app components they use in their applications (such as with simple obfuscation techniques). Consequently, if DREBIN relies on these features to classify the malware, attackers can easily bypass its detection. The same applies for the `Network addresses` category. Note that in their paper, DREBIN's authors stated that collecting the components' names may help identify the well-known components of malware. Although this may reveal relevancy within a specific time frame, we are concerned that changes in component names within variants will affect the classifier performance.

Analysing the features used in the previous section reveals that in our four datasets, at least 88.9% of the features belong to the `App components` and `Network addresses` categories. These are string features that are chosen by the developer (e.g., a class name) and identify elements that do not necessarily hold any semantic meaning beyond the scope of the sample. We refer to them as "id-features". Id-features are extracted from components (Activity, Service, Content Provider, Broadcast Receiver) names and `Network addresses`. Since these features are extracted from the apps of a specific dataset, they are ad hoc to that dataset. Indeed, an id-feature that is present in a dataset may not appear in another one, especially when malware evolves. In addition, id-features can be changed without affecting the behaviour of the apps. Consequently, a DREBIN classifier that considers id-features of a given dataset is likely to perform poorly outside this dataset.

[4]We recall that there are four types of components defined by the Android framework: Activity, Service, Broadcast Receiver, and Content Provider.

Table 3. Number of Id-Features Used in Our Replication of DREBIN

	2019_data	2018_data	2017_data	DREBIN_Like_data
# Features	1,230,854	1,331,583	1,486,191	389,957
# Id-Features	1,160,420	1,197,831	1,326,301	347,007
% Id-Features	94.28%	89.95%	89.24%	88.98%

We present in Table 3 the exact number of id-features in `2019_data`, `2018_data`, `2017_data`, and `DREBIN_Like_data`.

Finding 3

DREBIN's set of features contains a huge number of id-features. This raises a concern of generalisability if malware samples are identified based on learning non-generic features.

3 DISCRIMINATORY POWER OF DREBIN'S FEATURES

As we have established in the previous section, DREBIN uses a huge number of features to perform its task of prediction. Consequently, it is difficult to get insights about what DREBIN learns and on what basis it performs the prediction. Specifically, we cannot tell if DREBIN needs all these features for its task of detection nor what its main decisive features are. For a given Android app, we are curious to know to what extent DREBIN has captured its malware/benign behaviour using its set of features. This problem is our main motivation for this section, where we aim to reduce the number of features of our four classifiers while preserving a comparable performance.

3.1 What Are the Key Features That Enable the Prediction?

In this section, we conduct several experiments to analyse what the DREBIN classifier learns. Our aim is to create variations of DREBIN classifiers that use a small subset of the initial features and still perform well on the dataset. Analysing these features will make it possible to understand what this classifier captures, as well as the key features that direct its predictions.

Hypothesis 2

Some features can be removed with little to no loss in performance.

3.1.1 Method. We have developed our own custom feature selection approach to identify the smallest set of features that still yields reasonably high classification scores. Our feature selection technique unfolds the following four steps:

(1) We split each of our four datasets into subsets (80% for training and 20% for testing), and we train a DREBIN classifier using all the features (i.e., exactly what we have done in the previous section).
(2) With the feature ranking given by SVM, we train classifiers using the features that contribute the most in the prediction, starting with the most important feature and adding gradually the other features. Each time we add a feature, we compute the F1-score. This procedure is repeated until a subset of features yields a reasonably high F1-score. In our experimental setup, we consider an F1-score ≥ 0.8 to be reasonably high.
(3) We repeat steps (1) and (2) K-times, where we vary, each time, the training and testing sets. This step is necessary given that the feature set is dependent on the training set. At the

Table 4. Number of Red1 Features Sets and the Performance Scores

	2019_data	2018_data	2017_data	DREBIN_Like_data
# Features	19	156	158	188
Recall	0.928	0.764	0.782	0.725
Precision	0.888	0.877	0.921	0.948
F1-score	0.908	0.816	0.846	0.822

end of this step, we have K subsets of features, with each subset offering reasonably high performance.

(4) The union of the K sets of top features is further reduced by iteratively discarding features that can be removed without any loss in the performance. Therefore, at the end of this step, we have the minimal set of features that can achieve the same performance as the reference classifier trained with the whole union of the K sets.

We note that with our wrapper customised method, we do not aim to improve the prediction performance nor to optimise DREBIN's approach. Our aim is to create a classifier that reports good performance and uses a small number of features.

3.1.2 Results. We have applied our features reduction method from the previous section on the four datasets using $K = 10$. For each dataset, we have evaluated the performance of DREBIN using our reduced sets of features that we denote as Red1. Our evaluation is conducted using the 5-fold cross-validation technique. We present in Table 4 the number of features in Red1, for the four datasets, and the results of predictions: recall, precision, and F1-score.

As can be seen in Table 4, we were able to build DREBIN classifiers that, unlike DREBIN's replication, use only a small number of features with 2019_data, 2018_data, 2017_data, and DREBIN_Like_data datasets. These classifiers have been created in such a way that even if they use a small number of features, they still perform well and report high scores.

For 2019_data, we have been able to use 19 features to get a good F1-score (0.908), whereas with DREBIN replication, SVM uses 1,230,854 features to achieve an F1-score of 0.979.

For 2018_data, 156 features that represent again a very small subset of the features used in DREBIN replication (0.01%) are needed to recover 88.22% of the F1-score. Similar results have been achieved with 2017_data, where 158 features (0.01% of the initial features) have enabled the reporting of good scores (e.g., F1-score = 0.846), which recovers 91.16% of the initial F1-score. As for the DREBIN_Like_data, we were able to recover 87.08% of the F1-score using 0.048% of the initial features.

Finding 4

A significantly smaller subset of DREBIN features is enough to provide reasonable performance compared to the whole set of features. This suggests that most features in DREBIN are at best redundant and at worst useless.

We now assess to what extent the reduced sets of features are indeed relevant for the DREBIN classifier by investigating their information gain. The information gain of a feature represents the amount of information gained about the predicted class when observing this feature itself. The information gain can be used to verify if the reduced features sets indeed capture an important amount of information.

For each dataset, we have calculated the information gain of all the features, and we have ranked the features by their information gain. If a feature has a good ranking, it means that it captures more

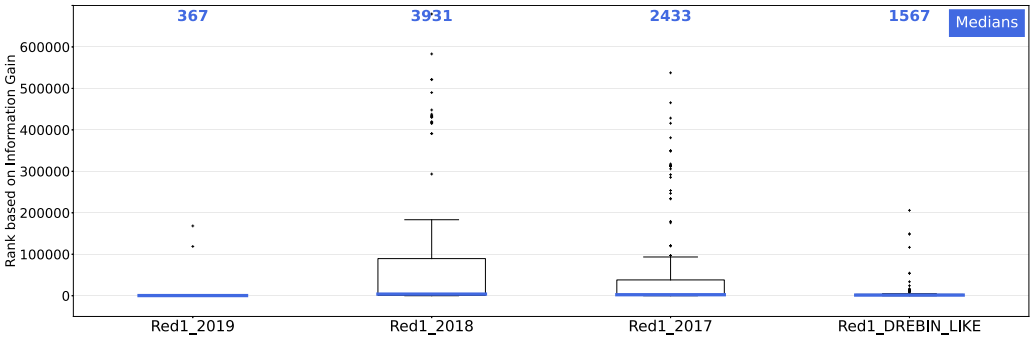

Fig. 1. Distribution of the rank of the features selected in the reduced features sets. Features are ranked according to their information gain for the four datasets. Numbers at the top of each distribution represent the median values.

information (than the other features). Consequently, the highly ranked features are important to the prediction. We present in Figure 1 the distribution of rankings (based on info gain values) of features included in the reduced features sets, and we represent their median values in blue colour. The distribution of rankings highlights that the features in the reduced sets are among those having the highest information gain.

3.1.3 Analysis. As shown previously, we were able to collect sets of features that not only are the most relevant features for the DREBIN classifier but also enable to report good prediction performance when they are used "alone" to train the classifier. Examining these features can lead to further insights about DREBIN's discriminative features.

Specifically, studying the distribution of the features in the dataset and their malicious/benign character is a potential lead to understand how the classifier works. In addition, analysing the features helps to validate DREBIN's capabilities, especially to confirm whether or not it learns general characteristics of goodware and malware apps.

We list in the following the 19 features present in the reduced features sets `Red1` of the `2019_data` dataset:

(1) `activitylist_com.silverbat.knightage.temmidlet`
(2) `activitylist_com.stub.stub08.appupdateactivity`
(3) `activitylist_com.twofloorhousedesign.lukoni.adsactivity`
(4) `activitylist_com.e4a.runtime.android.mainactivity`
(5) `restrictedapilist_android.telephony.telephonymanager.getline1number`
(6) `restrictedapilist_android.support.v4.view.accessibility.accessibilitynodeprovidercompat.performaction`
(7) `usedpermissionslist_android.permission.wake_lock`
(8) `suspiciousapilist_ljava/lang/runtime;->exec`
(9) `usedpermissionslist_android.permission.read_phone_state`
(10) `restrictedapilist_android.provider.settings$system.putstring`
(11) `usedpermissionslist_android.permission.get_accounts`
(12) `suspiciousapilist_landroid/telephony/telephonymanager.getsubscriberid`
(13) `restrictedapilist_android.media.mediaplayer.stop`
(14) `restrictedapilist_android.os.vibrator.cancel`
(15) `restrictedapilist_android.net.connectivitymanager.isactivenetworkmetered`

(16) requestedpermissionlist_android.permission.read_phone_state
(17) restrictedapilist_android.accounts.accountmanager.getaccounts
(18) suspiciousapilist_landroid/view/keyevent.getdeviceid
(19) restrictedapilist_android.media.audiorecord.<init>

For 2019_data, we can see that the reduced features sets contain four id-features (from the App components category), which are features numbers 1, 2, 3, and 4. The other features are not id-features, and they can give insights about some activities the app performs.

As for the reduced features sets of 2018_data, 2017_data, and DREBIN_Like_data, they also contain id-features that belong to both App components and Network addresses categories. Specifically, the id-features represent 61%, 60%, and 77% of Red1 for 2018_data, 2017_data , and DREBIN_Like_data, respectively. We remind that Red1 is the selection of the most important features of DREBIN.

The good results of DREBIN may be explained by code reuse among malware, coupled to the significant presence of id-features that excel at capturing elements shared across apps. Specifically, when the code of a malware app is reused, the attacker may not change the app components features set (i.e., activity names for example), which results in DREBIN predicting an application as malware because it contains the same component name of the app from which the code is reused (but also because of the presence of other relevant features when they are present in the app).

To illustrate the impact of id-features on the prediction, we have examined a malware APK[5] that contains one id-feature from Red1 (i.e., activitylist_com.silverbat.knightage.temmidlet). This application is predicted by DREBIN as malware. However, when we make a slight modification in the app by changing the name of this activity to another arbitrary name, the app escapes the detection of DREBIN. Note that changing the name of this id-feature does not impact the actual functioning of the app. It is a simple name that is chosen by the developer (or the attacker). Consequently, malware apps can easily evade the detection by changing the names of the id-features that are considered relevant by DREBIN, which raises a concern of generalisability.

Finding 5

A significant part of DREBIN's most relevant features are id-features.

Our method allows to identify the most important features that are related to the dataset at hand. The number and the composition of the features that we have collected differ from one dataset to another. This situation makes the reduction of feature sets dependent to the datasets, as we have noted with the four datasets (cf. Table 4).

3.2 Are All the Features Needed to Achieve Similar Results?

The performance achieved using our proposed reduced set of features is inferior to the full DREBIN. We are curious to know how many features are effectively needed to achieve similar performance.

Hypothesis 3

Only a subset of the features is needed to reach the performance reported when all the features are used.

[5]6013EDCCE42F8B0548AD750682BCB08C.

Table 5. Number of Optimised Features for Red2 and the Performance Scores

		2019_data	2018_data	2017_data	DREBIN_Like_data
Red2	**# Features**	73	1,183	241	296
	Recall	0.969	0.92	0.892	0.894
	Precision	0.975	0.929	0.945	0.978
	F1-score	0.972	0.925	0.918	0.934
DREBIN Replication F1-score (cf. Table 2)		0.979	0.925	0.928	0.944
# of Features		1,230,854	1,331,583	1,486,191	389,957
Red2 % Features Reduction		99.99%	99.91%	99.98%	99.92%

3.2.1 Method. For this experiment, we select the features based on the information gain as follows:

(1) We split each of our four datasets into subsets (80% for training and 20% for testing), and we calculate the F1-score using all the features (the ones constructed based on the training set).
(2) We calculate the information gain of all the features and rank them in descending order.
(3) We train SVM classifiers with these features starting with the one feature that has the highest rank and adding gradually the other features. Each time we add a feature, we calculate the F1-score. This procedure is repeated until the difference between this F1-score and the one obtained in step (1) is smaller than 0.01.
(4) The features from the previous step are further reduced by iteratively discarding features that can be removed without any loss in the performance. At the end of this step, we have the minimal set of features that can achieve the same performance as our replicated DREBIN. The resulted features are denoted as Red2.

3.2.2 Results. We apply the previous method to our four datasets, and we report the results in Table 5. We note that in this setup, it is not possible to use the cross-validation technique since we aim to compare the features and the scores of two dependent classifiers: one uses all the features (that are specific and depend on the training data), and the other classifier is trained with features that are an optimisation of the first classifier's features.

Specifically, our results in Table 5 are compared to the results in Table 2.

Overall, the results demonstrate that only a small set of features is needed to report scores that are as high as the scores achieved with all the features. We were able to reduce the initial features set by at least 99.9% for 2019_data, 2017_data, and DREBIN_Like_data, respectively. As for 2018_data, we observe that the number of features in Red2 is higher. This can be explained by the diversity and the composition of this dataset (i.e., it contains the highest number of goodware apps).

Finding 6

Only a small sets of features are needed to achieve results that are as excellent as the results reported when all the features are used.

We remind the reader that our aim through this investigation is not to replace DREBIN's features with those that we collect. Instead, we aim to identify DREBIN's most relevant features—that is, those that are necessary to its predictions. Such features can be further analysed to get insights about DREBIN's capabilities.

Table 6. Performance Scores of ONE-Feature DREBIN (i.e., the Feature That Has the Highest Info Gain)

	2019_data	2018_data	2017_data	DREBIN_Like_data
# Features	1	1	1	1
Recall	0.827	0.614	0	0
Precision	0.765	0.622	0	0
F1-score	0.795	0.618	0	0

3.2.3 The Impact of DREBIN's Features on the Learning Process. To get insights about how the performance of DREBIN evolves with the number of features, we provide in Figure 2 the plots that show the impact of adding gradually the features ranked with the information gain on the prediction. Each dataset is split into two subsets (80% for training and 20% for testing). We calculate the recall, the precision, and the F1-score starting with the most important feature and adding gradually the other features based on their information gain rank. For each dataset, the first graph represents the plots of recall, precision, and F1-score as a function of the number of ranked features used in the prediction. The plots in the right are a zoom of the first plots on recall, precision, and F1-score using the first 4,000 features.

As we can see from the first plots, the performance of prediction is generally stable after the first highly ranked features for the four datasets. The recall and the precision may increase at a certain time (after adding more features), but they evolve inversely. The F1-score metric that captures both the recall and the precision is almost stable when adding more features. The zoom plots confirm this statement since with the top 4,000 highly ranked features, the performance metrics are generally high and they are close to the recall_all, precision_all, and f1_all that we present in the plots with straight lines. These metrics are calculated using all the features from the training datasets.

These results suggest that DREBIN is notably impacted by a small number of dominant features that have the highest info gain. However, it is hardly affected by a huge number of features that have less information gain. We also observe that the detection performance does not monotonically increase with the addition of features. Indeed, for three of our datasets, there is actually a drop in performance after a number of features are included, and it can take up to 1 million additional features to come back to and marginally improve the performance obtained with the Red2 features.

Finding 7

A huge number of features have a minor influence on the prediction. Furthermore, adding subsets of features can even result in a performance decrease.

3.2.4 ONE-Feature DREBIN. In the previous section, we have seen that many features have a minor impact on DREBIN's prediction. Based on Figure 2, we also observe that a single feature (i.e., the first feature that has the highest information gain) enables DREBIN to yield a high detection performance for 2019_data and 2018_data. As for 2017_data and DREBIN_Like_data, the detection performance of DREBIN using the single feature is null. We present in Table 6 the prediction scores of DREBIN trained using the first feature that has the highest info gain and evaluated with 5-fold cross validation.

We observe that the overall performance of DREBIN using this ONE feature is surprisingly good for 2019_data and 2018_data. For both of these two datasets, the ONE feature is from DREBIN's "Suspicious API Calls" features set, which contains API calls that have access to sensitive data or

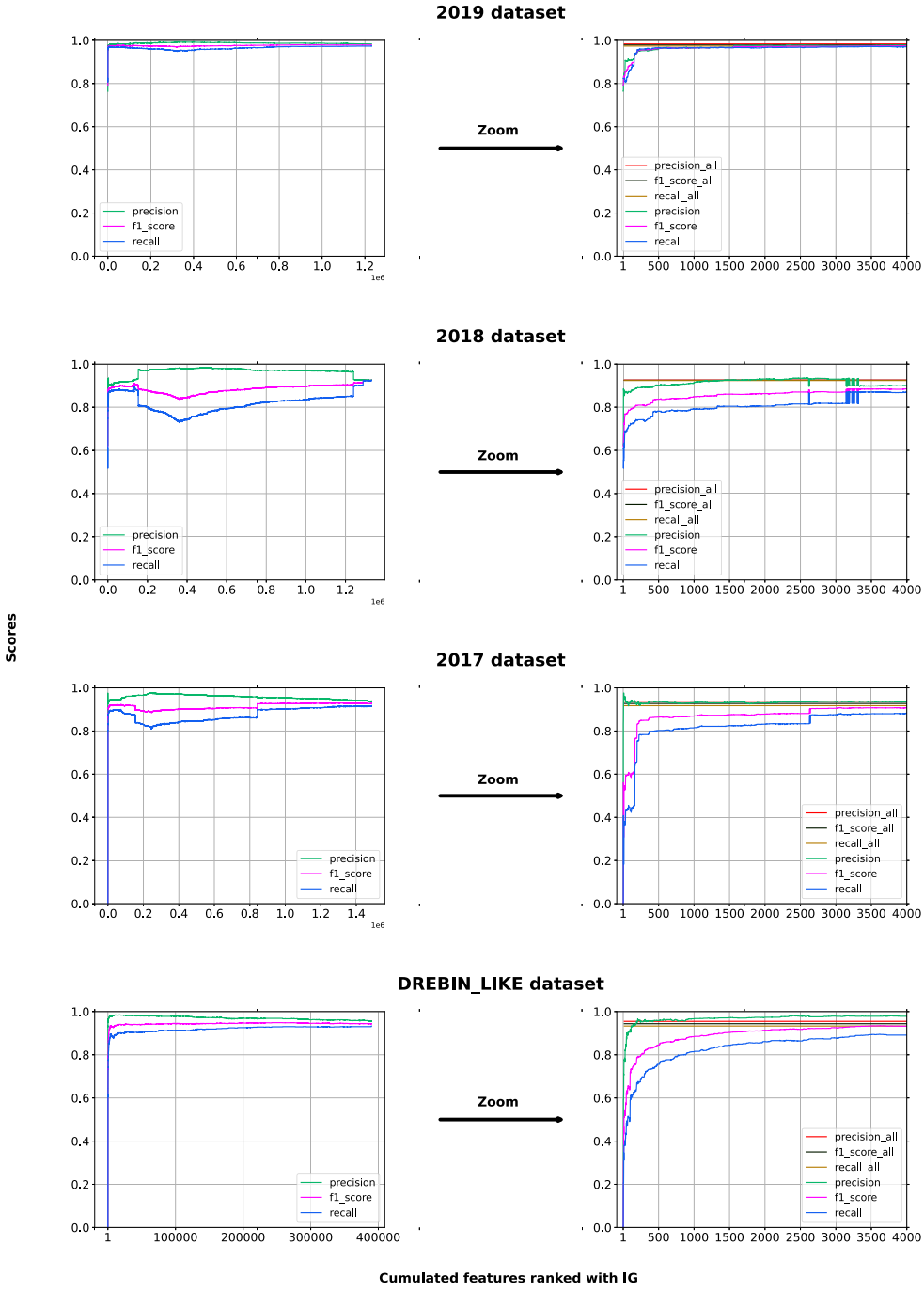

Fig. 2. The performance of DREBIN using the cumulated features ranked with info gain.

resources. This feature is landroid/content/context.getsystemservice. After inspecting this feature's presence in 2019_data apps, we found that it is present in 17.3% of the malware and 97.7% of the benign apps. In 2018_data, this feature exists in 38.6% malware and 98% benign apps. In these two datasets, the ONE feature is present in a large majority of the benign apps, and it is less frequent in malware apps (i.e., it is present in less than 50% of the apps).

Concerning 2017_data and DREBIN_Like_data, their ONE feature is restrictedapilist_android.support.v4.view.accessibility.accessibilitynodeprovidercompat.performaction and requestedpermissionlist_android.permission.send_sms, respectively. The ONE feature of 2017_data is present in 8.8% of malware and 69.6% of goodware. As for DREBIN_Like_data ONE feature, it exists in 53.4% of malware and 4.3% of benign apps. For these two datasets, the performance of DREBIN using the ONE feature is null.

We have also tested the ONE feature of 2019_data and 2018_data on both 2017_data and DREBIN_Like_data, but the detection performance is again null. The examination of this feature's presence in 2017_data and DREBIN_Like_data reveals that it is present in 75.1% and 81.6% of their malware apps and in 96.9% and 72% of their benign apps, respectively. This observation suggests that when DREBIN is trained using a feature that is predominant in goodware apps (or malware apps), it tends to associate its presence with the benign class (or the malware class).

Finding 8

A single feature can offer a surprisingly high detection rate.
An analysis of features is therefore necessary to assess their genericity (for detecting a variety of malware variants), as well as their discriminative power (for leading to accurate prediction).

4 ANALYSIS OF DREBIN CLASSIFIER EXPLANATIONS

In this section, we aim to perform an in-depth analysis of DREBIN's explanations to assess how well this approach explains the prediction (Section 4.2), and how consistent the explanations given by DREBIN are to malware samples of the same family (Section 4.3). Before diving into the analysis, in Section 4.1 we first present an overview of the malware families present in our datasets.

4.1 An Overview of Malware Families

In this section, we present and study the distribution of malware families in our four datasets. For each malware sample, we collected from VirusTotal the detection reports provided by the hosted antivirus engines. Then, we leveraged AVCLASS [40] to infer a unique malware family label for each sample. We present in Figure 3 the distribution of malware families in 2019_data, 2018_data, 2017_data, and DREBIN_Like_data as provided by AVCLASS.

We notice that we have a dominant malware family in three datasets. The family (jiagu) represents 78.6%, 50.24%, and 28.29% of the malware in 2019_data, 2018_data, and 2017_data, respectively. The difference of the distribution of this malware family in the three datasets can explain the very good results we were able to achieve for 2019_data. Indeed, in this specific dataset, around 80% of malware samples are from the same family. We postulate that by classifying these samples correctly, DREBIN can exhibit a very good recall. With this dataset, we were also able to retrieve a small number of features either to get insights about DREBIN's learning or to report similar scores as when all the features are used. The other datasets required more features to perform similar performance, potentially due to the diversity of apps in these datasets. For our next experiments, we will limit ourselves to DREBIN_Like_data malware since this dataset is the most diverse in terms of represented malware families.

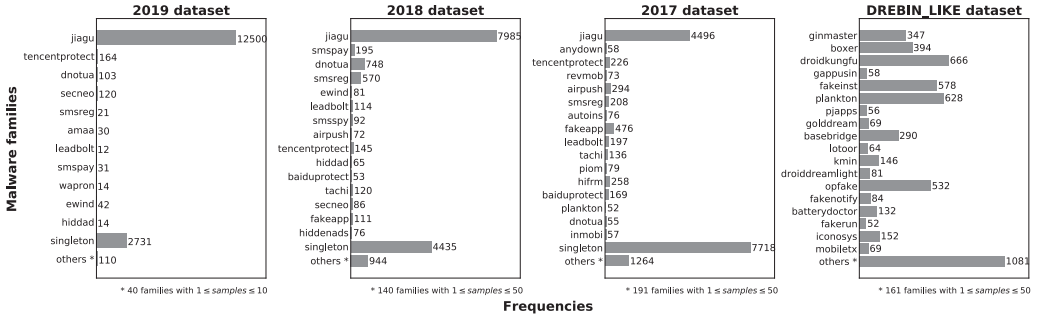

Fig. 3. The distribution of malware families in the datasets.

4.2 How Informative Are the Explanations Given by DREBIN?

In this section, we aim to assess the quality of the explanations provided by the DREBIN classifier. Specifically, we examine whether the information contained in these explanations is complete and adequately explains the prediction. We use "positive features" to refer to the features that have a positive SVM weight. DREBIN suggests that the top k positive features can be used to explain the prediction of malware applications. We formulate the following hypothesis.

Hypothesis 4

DREBIN's top positive features are sufficiently informative.

To verify our hypothesis, we examine two components:

(1) The contribution of the top positive features represented by their SVM weights;
(2) The number of the top positive features to use in the explanation.

4.2.1 The Contribution of DREBIN Top Positive Features. One key contribution of DREBIN is its ability to explain the detection by outputting the features that have the highest SVM weights after applying its linear transformation.

An example of DREBIN's top positive feature of a malware app that belongs to "droidkungfu" family in `DREBIN_Like_data` is *(0.966, 'intentfilterlist_android.intent.action.sig_str')*. In this example, `0.966` represents the SVM weight associated with the feature `'intentfilterlist_android.intent.action.sig_str'`.

DREBIN provides top k (with $k = 5$ in the experiments of the original paper) features that are relevant to the prediction, with their weight values. However, this information may actually not be sufficient to infer to what extent each feature was actually important (in terms of overall contribution for the detection, in comparison with all other features). Indeed, the contribution of the top five features may still be lower than the contribution of all other features combined. For instance, the value of 0.966 in this example does not reflect how powerful `'intentfilterlist_android.intent.action.sig_str'` is compared to the other positive features. The raw SVM weights suggested by DREBIN do not then quantify the importance of each individual feature to the prediction.

Finding 9

DREBIN's explanation does not reflect how much the features contribute to the prediction.

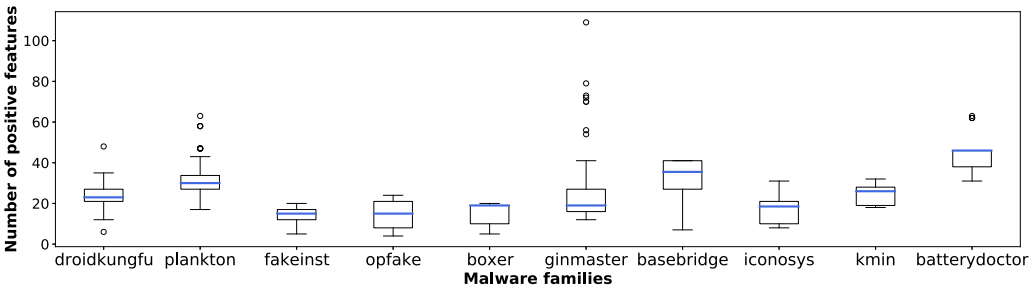

Fig. 4. The distribution of the number of positive features in the malware of the top 10 families of DREBIN_Like_data.

4.2.2 The Number of DREBIN's Top Positive Features. Another aspect we aim to investigate about DREBIN's explainability is how many top positive features are needed to explain the prediction. DREBIN's publication suggests that Android apps can be explained using the top k features, but it does not specify how to choose this number. Nonetheless, its evaluation experiments have been conducted using $k = 5$ top positive features for each family.

To verify if a specific number of the top positive features can be fixed to explain the prediction of the apps, we examine the distribution of the positive features in the apps of the top 10 malware families of DREBIN_Like_data.

Specifically, if the number of the positive features varies remarkably between the apps in general and between samples of the same family in particular, it would be difficult to decide about the threshold to fix for this number.

We provide in Figure 4 the number of all the positive features in the malware of the top 10 families of DREBIN_Like_data.

We notice that the number of all positive features differs remarkably from one app to another, and it is even different between apps that belong to the same family. This number ranges from 4 to more than 100 features in the malware of the DREBIN_Like_data test set. If this number is fixed at 5, we are confident that the top five features adequately explain the apps containing a small number of positive features. However, the top five features might fail to capture the primary features that manipulate the prediction of the apps that contain a large number of positive features, especially if the positive features have similar contributions. This problem is accentuated by the fact that the contribution of the features is not determined. For instance, if an application contains many positive features, each with a small contribution, we will not be able to decide about the threshold to fix for k to explain the prediction.

Finding 10

The number of positive features differs from one app to another and within apps of the same family, making it difficult to fix a threshold for the top positive features that explain the prediction.

4.3 How Consistent Are the Explanations Given to Samples of the Same Malware Family?

After showing that the number of positive features differs between samples of the same malware family, we seek to investigate the consistency of the top positive features within the same family

using DREBIN_Like_data. In our experiments, we use top five positive features to explain the prediction of the apps as it is suggested in DREBIN's paper.

Hypothesis 5

Malware samples of the same family are generally explained with the same features.

To verify our hypothesis, we visualise the distribution of the explanations in each of the top 10 families of DREBIN_Like_data. We plot for each malware sample the top 5 positive features that explain its prediction. Figure 5 shows the distribution of these explanations. For each malware sample, we plot the top five positive features with different colours to distinguish their rank (in explaining the prediction of the app). The interpretation of the five colours is explained in the figure. Each row represents a positive feature that has been used to explain at least one app that belongs to the family, and each column represents a malware sample. Ideally, we should see five straight lines with the following ordering of colours from top to bottom: red, blue, green, pink, and brown. This ideal graph reflects that the same five features are used to explain malware apps that belong to the same family. A line of the same colour shows that a feature have the same importance in the prediction of all the malware that belong to the family. The visual lecture of Figure 5 suggests that some families use the same features to explain most of their apps (e.g., "droidkungfu" and "plankton" families). However, the overlapping colours in the sub-graphs show inconsistency of explanations.

We have also calculated the overlap between the top five positive features in the malware samples of the 10 families. We provide in Figure 6 the distribution of the results.

We notice that for some families (e.g., basebridge and batterydoctor), there is a significant overlap of top features across samples in the family: there is a consistency of explanations for the majority of samples in the family. In some other families (e.g., fakeinst and opfake), however, the top features are not consistent across family samples. These findings suggest that top features that are used for explanations are not systematically sufficient to characterise a malware family.

We further report in Table 7 the total number of distinct top five features used in the top 10 malware families of DREBIN_Like_data. For each family, we provide the number of samples (found in the test set) and the total number of distinct features used to explain its malware samples (number of rows in Figure 5 visualisations). We notice that the number of distinct top five features used to explain the families' samples clearly exceeds five features. This number is generally higher when there are more samples in the malware family. The numbers in Table 7 suggest that DREBIN refers to a variety of features to "explain" samples within the same family.

Finding 11

Not all malware families can be characterised by the explanations provided by DREBIN.

5 ASSESSMENT OF DREBIN'S LEARNING POTENTIAL

In this section, we aim to evaluate DREBIN's ability to capture the concept of malware families, both within the family (i.e., if it is able to assemble malware samples that belong to the same family), and across families (i.e., if it separates malware that belong to different families).

5.1 Does DREBIN Capture the Concept of Malware Family (within the Family)?

To answer our question, we represent malware samples in DREBIN_Like_data by their embedded vectors, and we make the following hypothesis.

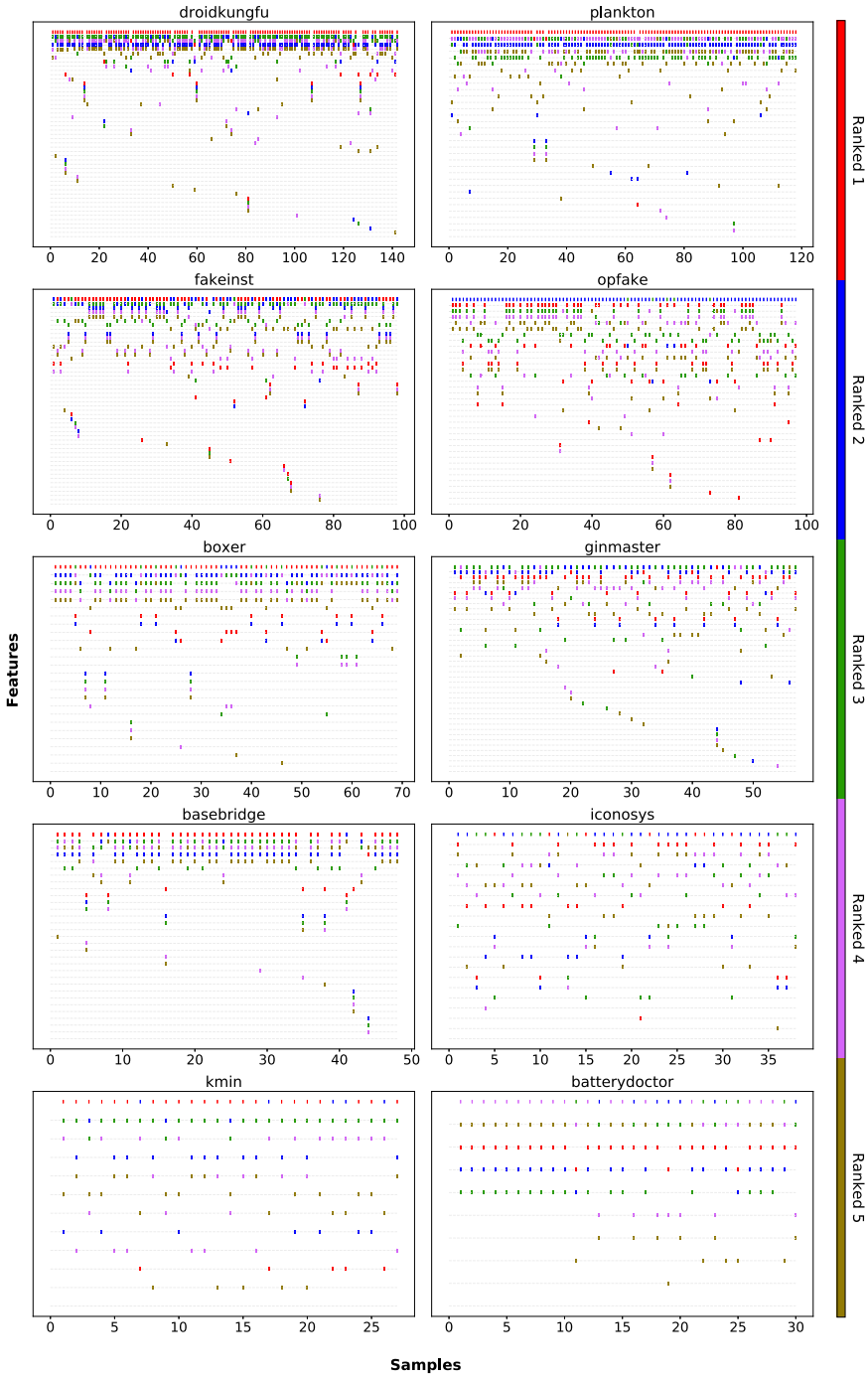

Fig. 5. The distribution of the top five features in the top 10 families of DREBIN_Like_data.

Table 7. Total Number of Distinct Top Features in the Top 10 Malware Families of DREBIN_Like_data

Families	No. of Samples	No. of Features
droidkungfu	142	48
plankton	118	32
fakeinst	98	48
opfake	97	35
boxer	69	25
ginmaster	57	39
basebridge	48	30
iconosys	38	20
kmin	27	11
batterydoctor	30	9

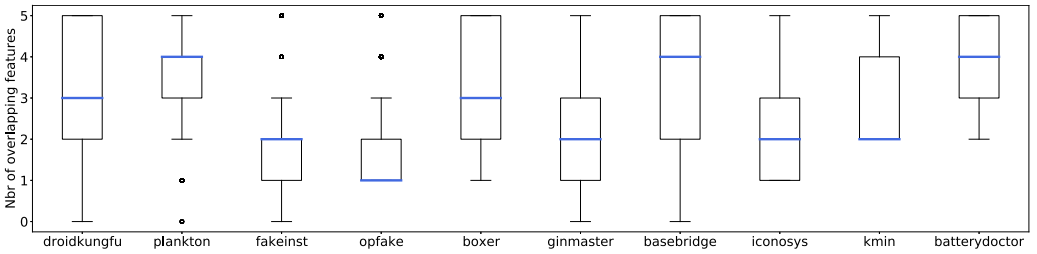

Fig. 6. The distribution of the overlapping top five features in the top 10 families of DREBIN_Like_data.

Hypothesis 6

In the n-dimensional space, DREBIN assembles malware samples of the same family.

To verify our hypothesis, we rely on the Euclidean distance between feature vectors to examine if DREBIN's features allow assembling of malware samples that belong to the same family in the n-dimensional space. Our aim is to compare the distribution of the distances and verify if malware samples that belong to the same family have smaller distances (i.e., they are close to each other in the n-dimensional space) compared to those from the 10 families combined. Specifically, for each family, we compare the Euclidean distance within the family and within the top 10 families of DREBIN_Like_data combined.

We plot in Figure 7 the distribution of the distances within the top 10 families combined, which we denote as "all", and the distribution of the distance within each of the 10 families from DREBIN_Like_data, which we denote as "family".

As we can see in Figure 7, the average Euclidean distances within a family are always smaller than the distances calculated using malware from the 10 families combined.

We have further compared the distribution of the distances within a family and within the top 10 families combined using the Mann-Whitney-Wilcoxon test [29, 46]. This test is used to verify whether two data distributions are identical.

For each of the top 10 families, we calculate the p-value of the Mann-Whitney-Wilcoxon test using the distances within the family and the distances within the 10 families combined. In the 10

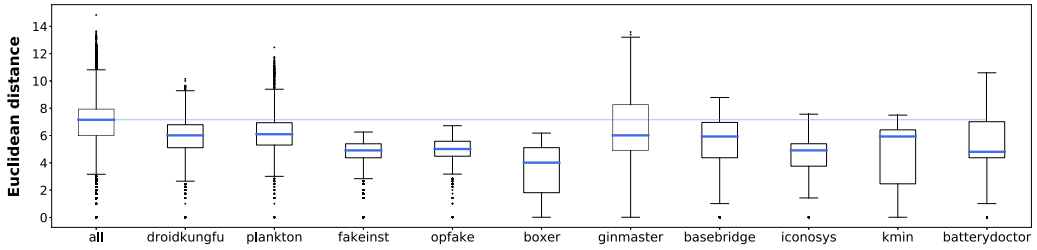

Fig. 7. The distribution of the distances within a family versus the distribution of the distances within all 10 families of DREBIN_Like_data combined.

Mann-Whitney-Wilcoxon tests, the p-value[6] is smaller than $9.42e^{-59}$. This result indicates that the null hypothesis is rejected and the difference between the data distributions is statistically significant. Therefore, we can conclude that the distributions of the distances within a family and within the 10 families combined are different.

Finding 12

DREBIN's features indeed offer a representation of apps that allows grouping together malware apps that belong to the same family.

5.2 Does DREBIN Capture the Concept of Malware Family (across Families)?

After evaluating DREBIN's ability to assemble malware samples within the same family, we seek to investigate if, in its feature space, DREBIN separates samples from different families.

Hypothesis 7

In the n-dimensional space, DREBIN separates malware apps of different families.

To verify our hypothesis, we compute, for each of the top 10 malware families in DREBIN_Like_data, the Euclidean distance within the family (similarly to Section 5.1) and across families. We denote $(F_i - F_j)$ as the distances across family i and family j for $(1 \leq i \leq 10)$ and $(1 \leq j \leq 10)$. Similarly to the previous section, we also calculate the distances within all top 10 families combined that we denote as "all". We provide in Figure 8 the distribution of these distances in the top 10 families. We draw the median of the distances within the family with a straight blue line to facilitate the comparison. Overall, the average distances within a family are smaller than the distances across families.

For each of the 10 top families, we have also compared the distribution of the distances within a family and across families using the Mann-Whitney-Wilcoxon test. We have conducted the test as explained in Section 5.1. For all the families, the p-value is smaller than $1.29e^{-10}$. The null hypothesis is then rejected, which indicates that the difference between the distributions of the distances within families and across families is statistically significant.

[6]The null hypothesis under this test assumes that the two distributions are indeed identical, and it is rejected if the p-value is smaller than $\alpha = 0.05$.

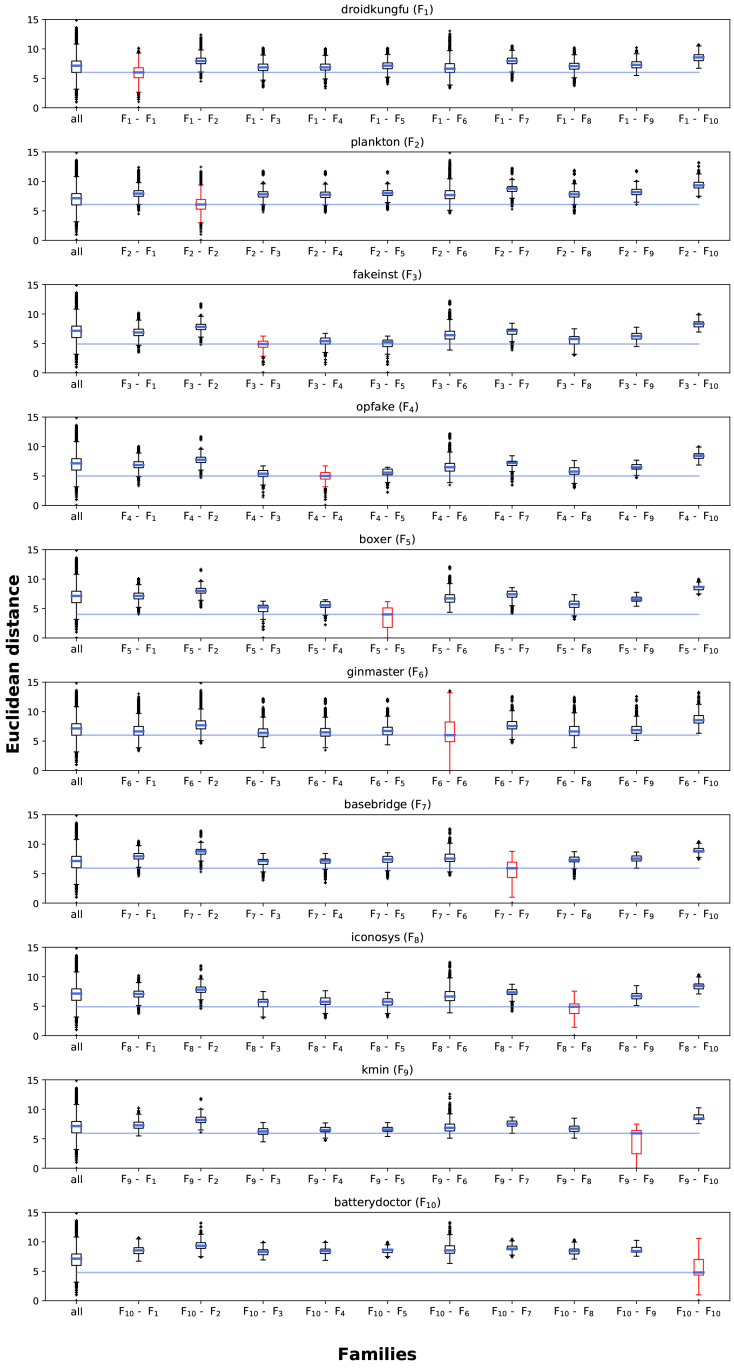

Fig. 8. The distribution of the Euclidean distances within families versus the distances across families in the top 10 families of `DREBIN_Like_data`.

Finding 13

DREBIN's representation enables the separation of samples that belong to different malware families.

6 DISCUSSION

Today, in the literature, a malware detection approach is considered to "work" when its assessment exhibits high detection scores. However, these scores have been proven to be biased by datasets [35] or evaluation methodology [3, 4]. In reality, practitioners expect more in-depth analysis that provides confidence or guarantees that approaches work. An example first step in this direction was implemented by DREBIN, which proposes explanations for its predictions, towards facilitating manual validation by analysts.

Given the importance of DREBIN as a literature milestone, we proposed to dive deep into its inner working to perform an investigation around its features, its explanations, and so on. Unfortunately, in the absence of guidelines in the literature, our explorations were guided by our past experiences and intuitions. Overall, our study findings are somewhat inconclusive, which highlights that the problem of answering "why it works" is a fundamentally difficult and unresolved problem.

6.1 Answering "Why It Works"

A large body of malware detection literature reports high performance scores using ML. Our in-depth analysis in this work revealed that it is hard to comprehensively characterise the reasons behind the good performance of a state-of-the-art classifier: (1) a substantial number of used features appear to be redundant, and (2) the detection explanations are not always consistent for samples in the same family.

Our first tentative to understand "why a classifier works" calls for a more focused research initiative around a framework that would provide tools and techniques to thoroughly assess malware detectors beyond detection scores. Such a framework would define the axes of analysis that each new malware detector needs to investigate beyond ML detection scores. The framework would suggest requirements and analyses that are crucial to gain insights into Android malware approaches' inner-workings, scope their properties, highlight their strengths, and uncover their limitations (or even pitfalls). The overall ambition is thus to ensure that researchers can associate their reported performance with specific key design choices in their contributions.

6.2 Implications of Our Findings

Our study revealed that many features which are sufficient to provide reasonable performance for the DREBIN classifier are actually id-features. We postulate that this may pose a generalisation problem. Our study further showed that many of the relevant features that contributed to the performance of DREBIN are indeed id-features.

We noted that a large proportion of malware can be detected using a single feature for two of our datasets, which further highlights both (1) the fundamental importance of evaluation datasets (in particular, of the diversity in these datasets), and (2) the necessity to assess whether features capture actual indicators of maliciousness or spurious correlations instead.

The consistency, or lack thereof, of top features for predicting samples in a same family raises questions on the relevance of DREBIN explanations.

7 RELATED WORK

Our selection of DREBIN is motivated by its notable impact on many research works that have compared against it (Section 7.1). Other studies have raised concern about the evaluation challenges (Section 7.2) and the biases that may affect ML-based malware experiments (Section 7.3).

7.1 The Impact of DREBIN

DREBIN is a well-known Android malware detector that has attracted researchers' attention since its publication in 2014. To evaluate their approaches, several authors have provided an experimental comparison with DREBIN. CASANDRA [33] is an online learning-based malware detector that has been evaluated against DREBIN, and a malware detector that relies on static analysis of the apps control flow graph [3]. CASANDRA's effectiveness is demonstrated using DREBIN's malware dataset and 5,000 randomly collected goodware applications. SIGPID [26] is a malware detector that is based on significant permissions features. Its authors have proposed an experimental comparison against DREBIN's approach and the PERMISSION-INDUCED RISK malware detector [45]. Many researchers have adopted a similar strategy by comparing against DREBIN (e.g., DroidOL [34], MalScan [47], RevealDroid [15], ASTROID [14], RepassDroid [48], TinyDroid [10], CDGDroid [49], DexRay [12], and a malware detector that relies on control flow graph and data flow graph of Android apps [50]).

Other researchers have chosen to develop their approaches by relying on DREBIN's malware dataset. DySign [23] is a fingerprinting technique for Android malware's dynamic behaviours that has been evaluated using malware samples from DREBIN dataset. APK Auditor [42] is a static analysis-based technique that characterises and classifies Android applications. Its authors have collected their evaluation dataset from DREBIN, Malware Genome Project [55], and Contagio mobile.[7] DREBIN's malware dataset has extensively contributed in the development of numerous research due to its availability (EC2 [9], AOMDroid [22], AspectDroid [2], a mobile botnet classification technique [53], StackDroid [37], a malware detector based on a Factorization Machine [24], and an Android malware family classifier [31]). Moreover, some researchers [17, 21] have studied the dataset itself.

DREBIN's approach has also been widely used in the context of adversarial attacks research. In 2017, DREBIN's approach was used to study the impact of adversarial attacks on linear and nonlinear classifiers [1]. In this study, the authors have demonstrated that a blind adversary can make DREBIN's performance drop by 88% when perturbing only 25% of the features. Another work [13] has proposed a learning algorithm to enhance linear models' security after showing that DREBIN's performance can deteriorate if skilled attackers manipulate it. Similarly, DREBIN has contributed to many other adversarial attacks research works [18–20, 25, 36, 43, 51], which proves that this classifier has a significant impact on the research community.

7.2 Evaluation Challenges

The evaluation of security systems has become a major concern in recent years. A recent study [44] has argued that the performance evaluation involves many operations and can never be fully expressed with a single number. Another study [41] has discussed the challenges in the intrusion detection approaches that rely on ML. Its authors argue that it is important to understand what the system is doing and what its capabilities and limitations are, insisting on the fact that the community does not benefit from trying some other combinations of ML algorithms and features sets with some known dataset.

[7] http://contagiodump.blogspot.com.

In malware experiments, researchers [38] have stressed the importance of performing prudent experimental evaluations to objectively assess the results. They have identified shortcomings after surveying papers from top-tier and less prominent venues, and they have proposed guidelines based on transparency, realism, correctness, and safety for prudent malware experiments.

Another work [39] has studied the impact of different factors on ML-based Android malware detectors' performance using their own ML approach. To study whether using more features always provides better results, the authors have investigated DREBIN's set of features (and the DroidSIFT [54] features set), and they have concluded that when removing the id-features, the performance of the classifier increases. Although the authors have not relied on the original implementation of DREBIN (i.e., DREBIN's features set + the K-NN algorithm), both their work and ours demonstrate that only a small subset of DREBIN's features set is needed to report a high detection performance. Moreover, we show that when the original set of features is used (i.e., id-features are included), DREBIN's most relevant features contain id-features.

7.3 Biases in ML-Based Malware Detection

In ML-based Android malware detection, many factors can influence or bias the results reported by the system, which makes its performance drop drastically when used in a real-world setting. Researchers [4, 35] have identified sources of experimental bias and proposed constraints to have a realistic evaluation. TESSERACT's authors [35] have used DREBIN and MaMaDroid [30] classifiers to motivate their finding, and showed that their performance drops significantly when evaluated with a real-world setting. They have also developed a framework that can be used to reveal the realistic performance of malware classifiers and eliminate the spatial and temporal biases.

Malware researchers widely rely on the cross-validation technique to evaluate the learning of the classifiers. However, it has been shown that although the performance of the classifier seems to be high when using this technique, the malware classifier performs poorly when used in the real-world setting [3]. Similarly, another study [32] has demonstrated that using the most recent training labels (e.g., the most recent reports from VirusTotal) that are in practice unavailable in real-world situations can inflate the performance of malware detectors by around 20%. Thus, the authors have introduced the temporal label consistency constraint that requires the training labels to be temporally precedent to the evaluation samples.

Recently, researchers [6] have determined 10 pitfalls that can introduce biases in the results reported by ML-based computer and network security systems. These pitfalls are related to dataset collection and labelling, system design and learning, performance assessment, and deployment. After studying the presence of these pitfalls in 30 papers from top-tier security venues, the authors have proposed a set of recommendations to develop sound ML-based security systems.

8 CONCLUSION

We have presented an exploratory analysis of the DREBIN malware detector with the aim to uncover insights about "how/why it works". Our study has extensively investigated the discriminatory power of the huge number of features that are used by DREBIN. We have thus identified DREBIN's most important features and showed that its discriminatory features contain id-features. We have also proposed a set of experiments both to evaluate the consistency of the explanations provided by DREBIN for malware samples of the same family and to assess DREBIN's ability to capture the concept of malware families.

Overall:

- DREBIN, given the stability of its performance across datasets, is a strong reference that researchers should consider when assessing the performance of malware detection approaches.

- Id-features are predominant within the feature set of DREBIN. Their importance in the decisions suggests that more research is needed on feature engineering that captures maliciousness in a more abstract (hence generalised) way. In the meantime, researchers should provide analysis on the presence of id-features and their impact on the reported performance. Further research should also investigate techniques for exploring id-features while mitigating their generalisation issues.
- The shortcomings of our first-step analyses suggest that the community needs further research on the design of quality metrics for classifiers beyond classical quantitative metrics of precision and recall.

Our work thus calls for more research into techniques and tools for analysing the performance of malware detection approaches. Although it is important to report high performance scores for state-of-the-art classifiers, the literature should also provide a principle-based systematic assessment into the inner-workings of the approach.

Malware detection today has become a critical concern that creates a challenge for managers, developers, and end users. Having tools that can demonstrate that a malware detector has a sound inner-working is thus paramount. Consequently, developing black-box systems with 100% performance scores in the lab does not inspire confidence anymore. Instead, more attention should be given to understanding how the systems work, and to what extent they are able to capture the malicious character of the apps so they can be trusted and deployed in real-world settings. Our work is an initial step for initiating a research roadmap into the assessment of malware detection classifiers. We hope that this research direction will be embraced by the community.

REFERENCES

[1] Z. Abaid, M. A. Kaafar, and S. Jha. 2017. Quantifying the impact of adversarial evasion attacks on machine learning based Android malware classifiers. In *Proceedings of the 2017 IEEE 16th International Symposium on Network Computing and Applications (NCA'17)*. 1–10. https://doi.org/10.1109/NCA.2017.8171381

[2] Aisha Ali-Gombe, Irfan Ahmed, Golden G. Richard, and Vassil Roussev. 2016. AspectDroid: Android app analysis system. In *Proceedings of the 6th ACM Conference on Data and Application Security and Privacy (CODASPY'16)*. ACM, New York, NY, 145–147. https://doi.org/10.1145/2857705.2857739

[3] Kevin Allix, Tegawendé F. Bissyandé, Quentin Jérome, Jacques Klein, Radu State, and Yves Le Traon. 2016. Empirical assessment of machine learning-based malware detectors for Android. *Empirical Software Engineering* 21, 1 (2016), 183–211. https://doi.org/10.1007/s10664-014-9352-6

[4] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Are your training datasets yet relevant? In *Engineering Secure Software and Systems*, Frank Piessens, Juan Caballero, and Nataliia Bielova (Eds.). Springer International Publishing, Cham, Switzerland, 51–67. https://doi.org/10.1007/978-3-319-15618-7_5

[5] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo: Collecting millions of Android apps for the research community. In *Proceedings of the 13th International Conference on Mining Software Repositories (MSR'16)*. ACM, New York, NY, 468–471. https://doi.org/10.1145/2901739.2903508

[6] Daniel Arp, Erwin Quiring, Feargus Pendlebury, Alexander Warnecke, Fabio Pierazzi, Christian Wressnegger, Lorenzo Cavallaro, and Konrad Rieck. 2020. Dos and don'ts of machine learning in computer security. *arXiv preprint arXiv:2010.09470* (2020).

[7] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, and Konrad Rieck. 2014. Drebin: Efficient and explainable detection of Android malware in your pocket. In *Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS'14)*.

[8] V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, and E. Bodden. 2015. Mining apps for abnormal usage of sensitive data. In *Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. 426–436. https://doi.org/10.1109/ICSE.2015.61

[9] T. Chakraborty, F. Pierazzi, and V. S. Subrahmanian. 2020. EC2: Ensemble clustering and classification for predicting Android malware families. *IEEE Transactions on Dependable and Secure Computing* 17, 2 (2020), 262–277.

[10] Tieming Chen, Qingyu Mao, Yimin Yang, Mingqi Lv, and Jianming Zhu. 2018. TinyDroid: A lightweight and efficient model for Android malware detection and classification. *Mobile Information Systems* 2018 (2018), Article 4157156.

[11] Nadia Daoudi, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2021. Lessons learnt on reproducibility in machine learning based Android malware detection. *Empirical Software Engineering* 26, 4 (2021), 1–53.

[12] Nadia Daoudi, Jordan Samhi, Abdoul Kader Kabore, Kevin Allix, Tegawendé F. Bissyandé, and Jacques Klein. 2021. DexRay: A simple, yet effective deep learning approach to Android malware detection based on image representation of bytecode. In *Proceedings of the 2nd International Workshop on Deployable Machine Learning for Security Defense (MLHat@KDD'21)*.

[13] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli. 2019. Yes, machine learning can be more secure! A case study on Android malware detection. *IEEE Transactions on Dependable and Secure Computing* 16, 4 (2019), 711–724.

[14] Yu Feng, Osbert Bastani, Ruben Martins, Isil Dillig, and Saswat Anand. 2016. Automated synthesis of semantic malware signatures using maximum satisfiability. *arXiv preprint arXiv:1608.06254* (2016).

[15] Joshua Garcia, Mahmoud Hammad, and Sam Malek. 2018. Lightweight, obfuscation-resilient detection and family identification of Android malware. *ACM Transactions on Software Engineering and Methodology* 26, 3 (2018), Article 11, 29 pages. https://doi.org/10.1145/3162625

[16] Hugo Gascon, Fabian Yamaguchi, Daniel Arp, and Konrad Rieck. 2013. Structural detection of Android malware using embedded call graphs. In *Proceedings of the 2013 ACM Workshop on Artificial Intelligence and Security (AISec'13)*. ACM, New York, NY, 45–54. https://doi.org/10.1145/2517312.2517315

[17] Hugo Gonzalez, Natalia Stakhanova, and Ali A. Ghorbani. 2015. DroidKin: Lightweight detection of Android apps similarity. In *International Conference on Security and Privacy in Communication Networks*, Jing Tian, Jiwu Jing, and Mudhakar Srivatsa (Eds.). Springer International Publishing, Cham, Switzerland, 436–453.

[18] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. 2017. On the (statistical) detection of adversarial examples. *arXiv preprint arXiv:1702.06280* (2017).

[19] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2016. Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435* (2016).

[20] Kathrin Grosse, Nicolas Papernot, Praveen Manoharan, Michael Backes, and Patrick McDaniel. 2017. Adversarial examples for malware detection. In *Computer Security—ESORICS 2017*, Simon N. Foley, Dieter Gollmann, and Einar Snekkenes (Eds.). Springer International Publishing, Cham, Switzerland, 62–79.

[21] Paul Irolla and Alexandre Dey. 2018. The duplication issue within the DREBIN dataset. *Journal of Computer Virology and Hacking Techniques* 14, 3 (2018), 245–249.

[22] Yu Jiang, Ruixuan Li, Junwei Tang, Ali Davanian, and Heng Yin. 2020. AOMDroid: Detecting obfuscation variants of Android malware using transfer learning. In *Proceedings of the International Conference on Security and Privacy in Communication Systems*. 242–253.

[23] E. B. Karbab, M. Debbabi, S. Alrabaee, and D. Mouheb. 2016. DySign: Dynamic fingerprinting for the automatic detection of Android malware. In *Proceedings of the 2016 11th International Conference on Malicious and Unwanted Software (MALWARE'16)*. 1–8.

[24] C. Li, K. Mills, D. Niu, R. Zhu, H. Zhang, and H. Kinawi. 2019. Android malware detection based on factorization machine. *IEEE Access* 7 (2019), 184008–184019. https://doi.org/10.1109/ACCESS.2019.2958927

[25] Deqiang Li, Qianmu Li, Yanfang Ye, and Shouhuai Xu. 2020. Enhancing deep neural networks against adversarial malware examples. *arXiv preprint arXiv:2004.07919* (2020).

[26] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye. 2018. Significant permission identification for machine-learning-based Android malware detection. *IEEE Transactions on Industrial Informatics* 14, 7 (2018), 3216–3225.

[27] Shixia Liu, Xiting Wang, Mengchen Liu, and Jun Zhu. 2017. Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics* 1, 1 (2017), 48–56. https://doi.org/10.1016/j.visinf.2017.01.006

[28] Arvind Mahindru and Paramvir Singh. 2017. Dynamic permissions based Android malware detection using machine learning techniques. In *Proceedings of the 10th Innovations in Software Engineering Conference (ISEC'17)*. ACM, New York, NY, 202–210. https://doi.org/10.1145/3021460.3021485

[29] Henry B. Mann and Donald R. Whitney. 1947. On a test of whether one of two random variables is stochastically larger than the other. *Annals of Mathematical Statistics* 18, 1 (1947), 50–60.

[30] Enrico Mariconti, Lucky Onwuzurike, Panagiotis Andriotis, Emiliano De Cristofaro, Gordon Ross, and Gianluca Stringhini. 2017. MaMaDroid: Detecting Android malware by building Markov chains of behavioral models. In *Proceedings of the ISOC Network and Distributed Systems Security Symposiym (NDSS'17)*.

[31] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni. 2017. Android malware family classification based on resource consumption over time. In *Proceedings of the 2017 12th International Conference on Malicious and Unwanted Software (MALWARE'17)*. 31–38. https://doi.org/10.1109/MALWARE.2017.8323954

[32] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, et al. 2016. Reviewer integration and performance measurement for malware detection. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Juan Caballero, Urko Zurutuza, and Ricardo J. Rodríguez (Eds.). Springer International Publishing, Cham, Switzerland, 122–141.

[33] A. Narayanan, M. Chandramohan, L. Chen, and Y. Liu. 2017. Context-aware, adaptive, and scalable Android malware detection through online learning. *IEEE Transactions on Emerging Topics in Computational Intelligence* 1, 3 (June 2017), 157–175. https://doi.org/10.1109/TETCI.2017.2699220

[34] A. Narayanan, L. Yang, L. Chen, and L. Jinliang. 2016. Adaptive and scalable Android malware detection through online learning. In *Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN'16)*. 2484–2491.

[35] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *Proceedings of the 28th USENIX Security Symposium (USENIX Security 19)*. 729–746. https://www.usenix.org/conference/usenixsecurity19/presentation/pendlebury.

[36] Fabio Pierazzi, Feargus Pendlebury, Jacopo Cortellazzi, and Lorenzo Cavallaro. 2020. Intriguing properties of adversarial ML attacks in the problem space. In *Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP'20)*. IEEE, Los Alamitos, CA, 1332–1349.

[37] Sheikh Shah Mohammad Motiur Rahman and Sanjit Kumar Saha. 2018. StackDroid: Evaluation of a multi-level approach for detecting the malware on Android using stacked generalization. In *Proceedings of the International Conference on Recent Trends in Image Processing and Pattern Recognition*. 611–623.

[38] C. Rossow, C. J. Dietrich, C. Grier, C. Kreibich, V. Paxson, N. Pohlmann, H. Bos, and M. V. Steen. 2012. Prudent practices for designing malware experiments: Status quo and outlook. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. 65–79. https://doi.org/10.1109/SP.2012.14

[39] Sankardas Roy, Jordan DeLoach, Yuping Li, Nic Herndon, Doina Caragea, Xinming Ou, Venkatesh Prasad Ranganath, Hongmin Li, and Nicolais Guevara. 2015. Experimental study with real-world data for Android app security analysis using machine learning. In *Proceedings of the 31st Annual Computer Security Applications Conference (ACSAC'15)*. ACM, New York, NY, 81–90. https://doi.org/10.1145/2818000.2818038

[40] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVCLASS: A tool for massive malware labeling. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses*. 230–253.

[41] R. Sommer and V. Paxson. 2010. Outside the closed world: On using machine learning for network intrusion detection. In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*. 305–316.

[42] Kabakus Abdullah Talha, Dogru Ibrahim Alper, and Cetin Aydin. 2015. APK auditor: Permission-based Android malware detection system. *Digital Investigation* 13 (2015), 1–14.

[43] Florian Tramèr, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. 2017. The space of transferable adversarial examples. *arXiv preprint arXiv:1704.03453* (2017).

[44] Erik van der Kouwe, Gernot Heiser, Dennis Andriesse, Herbert Bos, and Cristiano Giuffrida. 2019. SoK: Benchmarking flaws in systems security. In *Proceedings of the IEEE European Symposium on Security and Privacy (EuroS&P'19)*. IEEE, Los Alamitos, CA, 310–325. https://doi.org/10.1109/EuroSP.2019.00031

[45] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang. 2014. Exploring permission-induced risk in Android applications for malicious application detection. *IEEE Transactions on Information Forensics and Security* 9, 11 (2014), 1869–1882. https://doi.org/10.1109/TIFS.2014.2353996

[46] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics* 1, 6 (1945), 80–83.

[47] Y. Wu, X. Li, D. Zou, W. Yang, X. Zhang, and H. Jin. 2019. MalScan: Fast market-wide mobile malware scanning by social-network centrality analysis. In *Proceedings of the 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE'19)*. 139–150. https://doi.org/10.1109/ASE.2019.00023

[48] N. Xie, F. Zeng, X. Qin, Y. Zhang, M. Zhou, and C. Lv. 2018. RepassDroid: Automatic detection of Android malware based on essential permissions and semantic features of sensitive APIs. In *Proceedings of the 2018 International Symposium on Theoretical Aspects of Software Engineering (TASE'18)*. 52–59. https://doi.org/10.1109/TASE.2018.00015

[49] Zhiwu Xu, Kerong Ren, Shengchao Qin, and Florin Craciun. 2018. CDGDroid: Android malware detection based on deep learning using CFG and DFG. In *Proceedings of the International Conference on Formal Engineering Methods*. 177–193.

[50] Z. Xu, K. Ren, and F. Song. 2019. Android malware family classification and characterization using CFG and DFG. In *Proceedings of the 2019 International Symposium on Theoretical Aspects of Software Engineering (TASE'19)*. 49–56. https://doi.org/10.1109/TASE.2019.00-20

[51] Wei Yang, Deguang Kong, Tao Xie, and Carl A. Gunter. 2017. Malware detection in adversarial settings: Exploiting feature evolutions and confusions in Android apps. In *Proceedings of the 33rd Annual Computer Security Applications Conference (ACSAC'17)*. ACM, New York, NY, 288–302. https://doi.org/10.1145/3134600.3134642

[52] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-Sec: Deep learning in Android malware detection. *ACM SIGCOMM Computer Communication Review* 44, 4 (2014), 371–372. https://doi.org/10.1145/2619239.2631434

[53] M. Yusof, M. M. Saudi, and F. Ridzuan. 2017. A new mobile botnet classification based on permission and API calls. In *Proceedings of the 2017 7th International Conference on Emerging Security Technologies (EST'17)*. 122–127. https://doi.org/10.1109/EST.2017.8090410

[54] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao. 2014. Semantics-aware Android malware classification using weighted contextual API dependency graphs. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS'14)*. ACM, New York, NY, 1105–1116. https://doi.org/10.1145/2660267.2660359

[55] Y. Zhou and X. Jiang. 2012. Dissecting Android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*. 95–109. https://doi.org/10.1109/SP.2012.16

Received February 2021; revised September 2021; accepted November 2021