# Maat: Automatically Analyzing VirusTotal for Accurate Labeling and Effective Malware Detection

ALEIELDIN SALEM, SEBASTIAN BANESCU, and ALEXANDER PRETSCHNER,
Technische Universität München, Germany

The malware analysis and detection research community relies on the online platform `VirusTotal` to label Android apps based on the scan results of around 60 antiviral scanners. Unfortunately, there are no standards on how to best interpret the scan results acquired from `VirusTotal`, which leads to the utilization of different threshold-based labeling strategies (e.g., if 10 or more scanners deem an app malicious, it is considered malicious). While some of the utilized thresholds may be able to accurately approximate the ground truths of apps, the fact that `VirusTotal` changes the set and versions of the scanners it uses makes such thresholds unsustainable over time. We implemented a method, **Maat**, that tackles these issues of standardization and sustainability by automatically generating a **Machine Learning** (**ML**)-based labeling scheme, which outperforms threshold-based labeling strategies. Using the `VirusTotal` scan reports of 53K Android apps that span 1 year, we evaluated the applicability of **Maat**'s **Machine Learning** (**ML**)-based labeling strategies by comparing their performance against threshold-based strategies. We found that such **ML**-based strategies (a) can accurately and consistently label apps based on their `VirusTotal` scan reports, and (b) contribute to training **ML**-based detection methods that are more effective at classifying out-of-sample apps than their threshold-based counterparts.

CCS Concepts: • **Security and privacy** → **Malware and its mitigation**; *Mobile and wireless security;*

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

**ACM Reference format:**
Aleieldin Salem, Sebastian Banescu, and Alexander Pretschner. 2021. Maat: Automatically Analyzing VirusTotal for Accurate Labeling and Effective Malware Detection. *ACM Trans. Priv. Secur.* 24, 4, Article 25 (July 2021), 35 pages.
https://doi.org/10.1145/3465361



## 1 INTRODUCTION

A fundamental step in the process of implementing and evaluating novel malware detection methods is the creation of ground truths for the malicious and benign applications (hereafter apps) used to train those methods by accurately labeling them as malicious and benign. Inaccurate labels might impact the reliability of studies that inspect trends adopted by malicious apps and, more importantly, might impede the development of effective detection methods [9, 10, 12, 18]. Manual analysis and labeling of apps is presumed to be a reliable labeling method. However, it cannot cope

Authors' address: A. Salem, S. Banescu, and A. Pretschner, Technische Universität München, Boltzmannstraße 3, 85748 Garching bei München, Germany; emails: {salem, banescu, pretschn}@in.tum.de.

2471-2566/2021/07-ART25 $15.00
https://doi.org/10.1145/3465361

with the frequent release of malware. Moreover, some detection methods (e.g., **machine-learning** (**ML**)-based methods), require large numbers of data for training, which renders manual analysis and labeling infeasible. Consequently, researchers turn to online platforms, such as `VirusTotal` [38], that provide scan results from different commercial antiviral software [14, 36, 39, 41, 43].

The platform, `VirusTotal`, does not label apps as malicious and benign. Given a hash of an app or its executable, the platform returns the labels given by around 60 antiviral scanners to the app along with information about its content and runtime behavior. It is up to the platform's user to decide upon strategies to interpret such information to label apps as malicious and benign. Unfortunately, there are *no standard procedures* for interpreting the scan results acquired from `VirusTotal` to label apps. Researchers, hence, use their intuitions and adopt ad hoc threshold-based strategies to label the apps in the datasets used to train and evaluate their detection methods or they release to the research community as benchmarks. In essence, threshold-based labeling strategies deem an app as malicious if the number of antiviral scanners labeling the apps as malicious meets a certain threshold. For example, based on `VirusTotal`'s scan reports, Li et al. labeled the apps in their *Piggybacking* dataset as malicious if at least one scanner labeled them as malicious [14]. Pendlebury et al. labeled an app as malicious if four or more scanners did so, and based the evaluation of their tool on such threshold [7]. Wei et al. labeled apps in the *AMD* dataset as malicious if 50% or more of the total scanners labeled an app as such [41]. Finally, the authors of the *Drebin* dataset [3] labeled an app as malicious if at least 2 out of 10 scanners they manually selected courtesy of their reputation (e.g., `AVG`, `BitDefender`, `Kaspersky`, `McAfee`) did so.

Some of the aforementioned labeling strategies may indeed accurately label apps better than others. However, researchers have found `VirusTotal` to frequently change (e.g., by manipulating the scanners it uses in its scan reports) [18, 19, 23]. Changing the (number of) scanners in the scan reports means that threshold values that used to yield the most accurate labels at some point in time might change in the future as `VirusTotal` changes the scanners it includes in its scan reports. That is, `VirusTotal`'s dynamicity renders threshold-based labeling strategies that rely on fixed thresholds as **not sustainable**. To cope with the dynamicity of `VirusTotal`, researchers have to **manually** and **regularly** analyze `VirusTotal` scan reports to identify the current optimal thresholds to use in labeling the apps they use in their experiments, which is infeasible.

Using out-of-date or inaccurate thresholds alters the distribution of malicious and benign apps in the same dataset, effectively yielding different datasets and, in turn, different detection results as revealed by recent studies [7, 27]. On the one hand, researchers might dismiss promising detection approaches, because they underperform on a dataset that utilizes a labeling strategy that does not reflect the true nature of the apps in the dataset. On the other hand, developers of inadequate detection methods might get a false sense of confidence in the detection capabilities of their detection methods because they perform well, albeit using inaccurate labeling strategies [22, 28].

Until a more stable alternative to `VirusTotal` is introduced, the research community will continue to use `VirusTotal` to label apps using subjective thresholds. So, the overarching objective of this article is to provide the research community with actionable insights about `VirusTotal`, the aspects of its dynamicity, its limitations, and how to optimally interpret its scan reports to label Android apps accurately. To that end, we implemented a framework, **Maat**,[1] which is meant to provide the research community with an automated and a systematic procedure to infer the currently adequate labeling strategies to use against `VirusTotal`. **Maat** automatically analyzes

[1]Maat is the ancient Egyptian goddess of truth, balance, harmony, and justice. Our framework builds **ML**-based labeling strategies that harmonize the labels given by different `VirusTotal` scanners to provide accurate and reliable labels to apps.

a set of VirusTotal scan reports of pre-labeled apps to identify the set of (likely) correct and stable VirusTotal scanners at a given point in time. **Maat** then uses **ML** to build labeling strategies based on this and further information. **Maat** defines our technical contribution. Our methodological contribution then is a direct consequence: Whenever a new detection method is to be trained or evaluated, we suggest to apply **Maat** to the *most recent* VirusTotal scan reports, hence build the *currently best* labeling strategy and use this strategy to (re-)label existing collections of apps. However, unlike threshold-based labeling strategies, our experiments show that **Maat**'s **ML**-based labeling strategies can withstand VirusTotal's dynamicity for longer periods of time and, hence, **need not** be retrained on a regular basis.

**The contributions** of this article, therefore, are the following:

— Implementing and publicly releasing the code of **Maat** (Section 4): a framework that provides the research community with a systematic method to generate **ML**-based labeling strategies on-demand based on the current scan results provided by VirusTotal. The results of our experiments show that **Maat**'s **ML**-based labeling strategies are less sensitive to the dynamicity of VirusTotal, which enabled them to (a) accurately label apps based on their VirusTotal scan reports more consistently than their threshold-based counterparts and (b) improve the detection capabilities of **ML**-based detection methods (Section 5).

— Through the measurements and experiments conducted in this article, we found **four** main limitations of VirusTotal that sometimes undermine its reliability and usefulness. Those limitations are (a) refraining from frequently re-analyzing and re-scanning apps, (b) changing the set of scanners used to scan the same apps on a frequent basis, (c) using inadequate versions of scanners that are designed to detect malicious apps for other platforms, and (d) denying access to the history of scan reports (Section 6). We believe that revealing and demonstrating these limitations depicts one significant step toward designing platforms that mitigate VirusTotal's limitations.

— Unlike previous research efforts that are in pursuit of identifying a universal set of VirusTotal scanners that are more correct than others, in Section 4.2, we demonstrate that there is **no universal set** of scanners that can always accurately label apps in any dataset and at any time period. In fact, those two factors significantly alter the correct set of scanners based on Mohaisen and Alrawi's correctness score [19]. However, using **Maat**, researchers can automate the process of identifying the current set of VirusTotal scanners that can accurately label apps within a given time period and, in turn, help train reliable **ML**-based labeling strategies.

## 2 DATASETS

In this section, we briefly discuss the composition and the role of the datasets we used in motivating the need for **Maat** and for conducting experiments to evaluate it. Table 1 shows the datasets we use in this article.

The largest dataset we use in this article is a combination of 24,553 malicious apps from the *AMD* dataset [41] and 24,162 benign apps we downloaded from *AndroZoo* [2]. The malicious apps of *AMD* are meant to provide an overview of malicious behaviors that can be found in Android malware, spanning different malware families (e.g., DroidKungFu [45], Airpush [6], Dowgin [8]) and different malware types (e.g., Adware, Ransomware(ware), and Trojan). To build the dataset, the authors of *AMD* only considered apps whose VirusTotal scan reports indicate that at least 50% of the scanners deem them as malicious, clustered them—based on behavioral semantics—into a total of 71 malware families and 135 varieties, and manually analyzed three randomly selected samples of each variety to ensure the malignancy of apps. After analysis, the behavior of each

Table 1. The Datasets We Utilize in this Article: Their Composition, Their Sources, and Their Purposes Within This Article

| Dataset Name | Total Apps | Source | Usage |
|---|---|---|---|
| *AMD+GPlay* | 24,553 (malicious) 24,162 (benign) | *AMD*'s Website *AndroZoo*'s servers | ∘ Training **ML**-based labeling strategies (Section 4) ∘ Calculating scanner correctness (Section 4.2) |
| *AndroZoo* | 6,172 (unlabeled) | *AndroZoo*'s servers | ∘ Training **ML**-based detection methods (Section 5.2) |
| *Hand-Labeled* | 100 (76% benign+24% malicious) | *AndroZoo*'s servers | ∘ Demonstrating impact of VirusTotal's dynamicity on threshold-based labeling strategies (Section 3) ∘ Testing accuracy of labeling strategies (Section 5.1) ∘ Testing accuracy of **ML**-based detection methods (Section 5.2) |
| *Hand-Labeled 2019* | 100 (90% benign+10% malicious) | *AndroZoo*'s servers | ∘ Demonstrating impact of VirusTotal's dynamicity on threshold-based labeling strategies (Section 3) ∘ Testing accuracy of labeling strategies (Section 5.1) ∘ Testing accuracy of **ML**-based detection methods (Section 5.2) |

family is represented as a human-readable, graphical representation[2] of the behavior adopted by apps in each of the 135 malware families that can be found in the dataset. The involvement of human operators in labeling the apps and the high number of scanners deeming them malicious significantly decreases the likelihood of a benign app being mistakenly labeled as malicious. So, we consider all apps in the *AMD* dataset as malicious.

As for the benign apps we acquired from *AndroZoo*, we downloaded a total of 30,023 apps that were gathered from the Android official app store, *Google Play*. Google Play employs various checks to ensure the sanity of an app upon being uploaded [21], but sometimes malicious apps make it to the marketplace [15, 39]. So, we only considered apps whose VirusTotal scan reports indicate that **no** scanners deemed them as malicious during our evaluation period of **Maat** (i.e., between November 2018 and September 27th, 2019). This criterion does not guarantee that the apps' scan reports will continue to have a *positive* attribute of zero in the future. However, given that the apps were collected from Google Play, we presume that the VirusTotal scan reports of such apps will not radically change in the future. Consequently, we consider all apps in the *GPlay* dataset that fit the aforementioned criterion (i.e., 24,162 apps), as benign.

We refer to the combination of apps in the two previous datasets as *AMD+GPlay*, whose VirusTotal scan reports are used to train **Maat**'s **ML**-based labeling strategies. We downloaded the existing scan reports of those apps in November 2018 and found that the overwhelming majority of those apps were last scanned in 2018, albeit spread across different months. So, we reanalyzed all of those apps in November 2018. Between April 12th, 2019, and November 8th, 2019, we reanalyzed all of the 53K apps and downloaded the latest versions of their VirusTotal reports every 2 weeks in accordance with Kantchelian et al.'s recommendations [12]. Aware of the fact that some apps in the dataset are as old as 8 years, we attempted to acquire their older VirusTotal scan reports. Unfortunately, access to such reports is not available under academic licenses.

[2]Example: Airpush family's first variety (http://tiny.cc/34d86y).

The second dataset we use is a random collection of 6,172 apps developed between 2018 and 2019 and downloaded from *AndroZoo*. So, we refer to it as *AndroZoo* throughout this article. Unlike apps in *AMD+GPlay*, this dataset does not focus on a particular marketplace or class (e.g., malicious). This dataset is used in Section 5 to assess the ability of different labeling strategies to train more accurate detection methods. This dataset is meant to simulate the process of a researcher acquiring new Android apps and labeling them to train a **ML**-based detection method that detects novel Android malware, that `VirusTotal` scanners are yet to assign labels to. So, we use apps in this dataset to statically extract numerical features from their **Android Package** (**APK**) archives, represent them as feature vectors, and use them for training different types of ML classifiers. We use different threshold-based and **ML**-based labeling strategies to label those feature vectors prior to training the aforementioned classifiers.

The trained classifiers are then used to label apps in two small test datasets and compare the predicted labels with the ground truth. We refer to those small datasets as *Hand-Labeled*[3] and *Hand-Labeled 2019*.[4] Both datasets comprise 100 Android apps that were downloaded from *AndroZoo* and *manually analyzed and labeled*, to acquire reliable ground truth. The exact process we adopted in analyzing and labeling those apps can be found online. The primary difference between both datasets is that apps in the latter were developed in 2019. We ensured that apps in both datasets do not overlap with apps in the *AMD+GPlay* and *AndroZoo* datasets. We manually analyzed such 200 apps to acquire reliable ground truth that depicts the apps' true nature.

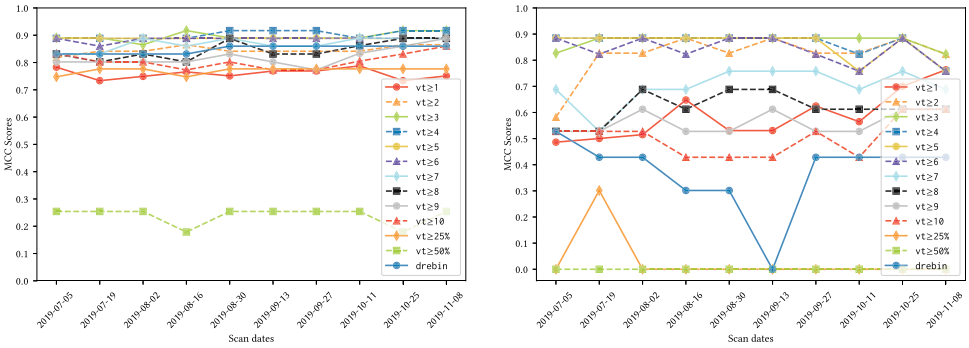
## 3 MOTIVATING EXAMPLES

During the evaluation of an Android malware detection method we implemented, we came across a dubious scenario involving a test app called `TP.LoanCalculator` that is part of the *Piggybacking* [14] dataset. Despite being labeled by the dataset authors as malicious, our detection method deemed this test app[5] as benign because it had the same metadata (e.g., package name and description), compiler, and even code base digest as one benign app[6] that our detection method kept in a repository of reference benign apps.

Apps in the *Piggybacking* dataset were labeled with the aid of `VirusTotal` scan reports [14]. After querying `VirusTotal` for the scan reports of both apps, we found that the test app was labeled malicious by 14 out of 60 antiviral software scanners, whereas all scanners deemed the reference app as benign, which coincides with the authors' labels. However, we noticed that the scan reports acquired from `VirusTotal` indicated that the apps were last analyzed in 2013. So, we submitted the apps' **Android Package** (**APK**) archives for re-analysis in November 2018 to see whether the number of scanners would differ. After re-analysis, the malicious test app had three more scanners deem it malicious. More importantly, the number of scanners deeming the benign reference app as malicious changed from zero to 17 after re-analysis. Ultimately, we found that the reference app initially labeled and released as part of the *Piggybacking* dataset as a benign app is, in fact, another version of a malicious app of the type `Adware`.

The authors of *Piggybacking* did not intentionally mislabel apps. The most likely scenario is that, at the time of releasing the dataset, the reference app was still deemed as benign by the `VirusTotal` scanners. To further demonstrate the impact of time on the scan reports of both apps, we re-analyzed them 6 months later (i.e., in April 2019), to check whether the number of `VirusTotal` scanners deeming them malicious changed. We found that the number of scanners

[3] http://tiny.cc/95bhaz.
[4] http://tiny.cc/a7bhaz.
[5] 2b44135f245a2bd104c4b50dc9df889dbd8bc79b.
[6] d8472cf8dcc98bc124bd5144bb2689785e245d83.




(a) *Hand-Labeled* (b) *Hand-Labeled 2019*

Fig. 1. The labeling accuracy of different threshold-based labeling strategies against apps in *Hand-Labeled* and *Hand-Labeled 2019* datasets based on their `VirusTotal` scan reports downloaded between July 5th, 2019 and November 8th, 2019. Accuracy is calculated in terms of the **Matthews Correlation Coefficient** (**MCC**) of each labeling strategy.

deeming the test and reference apps as malicious decreased from 17 to 11 and 10 scanners, respectively. Moreover, the total number of scanners respectively changed from 60 scanners to 59 and 57.

Another interesting fact is that the percentages of scanners deeming both apps as malicious are 18.64% and 19.3%, with some renowned scanners including `AVG`, `McAfee`, `Kaspersky`, `Microsoft`, and `TrendMicro` continuing to deem both apps as benign. According to the dataset authors' strategy to label an app as malicious if at least one scanner deems it so [14], both apps would be labeled as malicious. The same would not hold for the authors of the *AMD* dataset who consider an app as malicious if at least 50% of the `VirusTotal` scanners deem it malicious [41].

With this example, we wish to demonstrate the following issues with using `VirusTotal` to label Android apps. First, the lack of a standard method to interpret `VirusTotal`'s scan results encourages researchers to use different threshold-based labeling strategies to label apps, which might lead to completely different verdicts on the labels of the same apps. Assuming that the research community manages to standardize the thresholds used to label apps based on their `VirusTotal` scan reports, the second issue is that such scan reports are dynamic and continuously change over time due to either (a) scanners updating their verdicts on some apps or (b) the platform adding/removing scanners [18, 19]. This continuous change leads to scan report attributes, such as the number of scanners deeming an app as malicious (i.e., *positives*), to fluctuate. Consequently, thresholds that reflected the ground truth of an app accurately are also expected to change.

To demonstrate the fluctuations of `VirusTotal`'s scan reports and its attributes and how that impacts the performance of threshold-based labeling strategies, consider the plots in Figure 1. In this figure, we plot the performance of different threshold-based labeling strategies, including ones previously utilized within the literature, on the *Hand-Labeled* and *Hand-Labeled 2019* datasets between July 5th, 2019 and November 8th, 2019 in terms of **Matthews Correlation Coefficient** (**MCC**) score ($\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$). We use the **MCC** score to describe the performance of the classification models because conventional metrics, such as accuracy, are unable to capture or penalize bias toward certain classes in imbalanced datasets [5]. For example, using the accuracy metric ($\frac{TP+TN}{P+N}$), the $\texttt{vt} \geq 50\%$ would have a constant score of 0.90 for classifying apps in the *Hand-Labeled 2019*, effectively rewarding its bias toward the benign class that comprises the majority of apps in this dataset. However, **MCC** is capable of capturing this bias of such labeling

strategy toward classifying apps as benign and gives a value[7] that objectively describes the strategy's performance, namely, 0.0. Another metric that is widely used within the scienfitic community is the $F_1$ score ($\frac{TP}{TP+\frac{1}{2}(FP+FN)}$). In this example, the $F_1$ score would mimic that of **MCC** because the true-positive score ($TP$) is 0.0. However, unlike the **MCC** score, $F_1$ does not consider true negatives ($TN$) in its equation, which is significant to assess the performance of a classification model on the negative samples in the validation and test datasets (i.e., benign apps).

As seen in the figure, the labeling accuracy noticeably differs from one threshold-based labeling strategy to another. For example, researchers using $\texttt{vt} \geq 3$ or $\texttt{vt} \geq 4$ [7, 18] to label apps in the *Hand-Labeled 2019* dataset will get more accurate labels than those adopting $\texttt{vt} \geq 1$ [14] or $\texttt{vt} \geq 50\%$ [41]. Moreover, the **MCC** scores of such labeling strategies appear to fluctuate over a mere period of 4 months, especially against recently developed Android (malicious) apps in the *Hand-Labeled 2019* dataset. In conclusion, the dynamicity of `VirusTotal` means that fixed thresholds cannot be relied on for prolonged periods of time to label Android apps as malicious and benign. So, researchers have to analyze `VirusTotal` scan reports in order to identify the currently optimal threshold to use in the labeling process, which is indeed a time-consuming process. This process is further prolonged by the fact that `VirusTotal` does not automatically re-analyze the apps it stores in its repositories and relies on its users to manually initiate re-scans. Needless to say, the larger the datasets a researcher possesses, the longer it takes to re-scan all apps given the limitations on the number of **Application Programming Interface** (**API**) requests allowed per day for academic licenses.

**`VirusTotal` Limitation 1**

`VirusTotal` does not re-scan the apps it possesses on a regular basis and delegates this task to manual requests issued by its users. One direct consequence of this decision is prolonging the process of acquiring up-to-date scan reports of apps, especially under temporary academic licenses that grant users a limit of 20K **Application Programming Interface (API)** requests per day.

## 4 MAAT'S ML-BASED LABELING STRATEGIES

In the previous section, we demonstrated the subjectivity of threshold-based labeling strategies and their sensitivity to `VirusTotal`'s dynamicity and evolution. This dynamicity forces researchers to identify the current optimal thresholds, which might entail a process that is infeasible to perform on a regular basis. Our framework, **Maat**, is designed to address these problems as follows. First, to avoid the subjectivity of threshold-based labeling strategies, **Maat** provides the research community with a systematic method to analyze `VirusTotal` scan reports to devise labeling strategies. Second, **Maat**'s processes of analyzing `VirusTotal` scan reports and of devising **ML**-based labeling strategies are on-demand and fully automated. Third, the resulting **ML**-based labeling strategies do not rely on a fixed number of `VirusTotal` scanners to label apps as malicious and benign; instead, using two different types of features extracted from `VirusTotal` scan reports, **Maat** relies on **ML** algorithms to identify the currently correct and stable `VirusTotal` scanners, which makes them less susceptible to the dynamicity of `VirusTotal` and the changes it introduces to scan reports.

[7]The **MCC** values range from −1.0 (i.e., all apps were misclassified) to 1.0 (i.e., perfect classification), with the value of 0.0 indicating a classification ability no better than random classification.

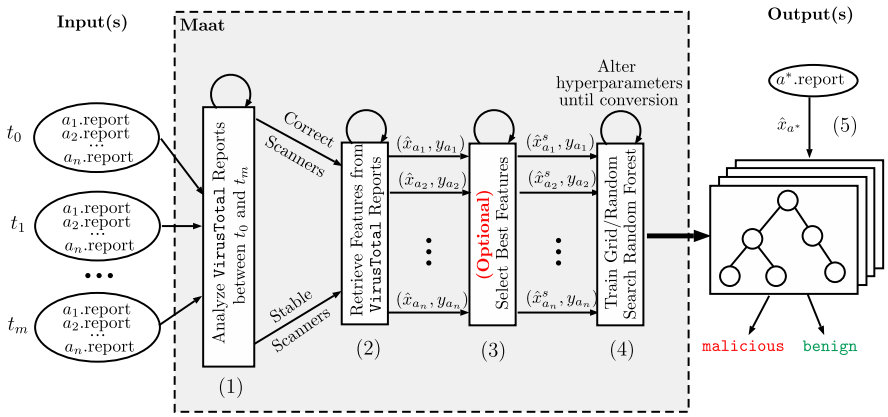



Fig. 2. The process adopted by **Maat** to construct ML-based labeling strategies by analyzing VirusTotal scan reports and training a random forest to label apps as malicious and benign according to their VirusTotal scan reports.

### 4.1 Overview

Prior to delving into the measurements and experiments we performed, we briefly discuss how our framework, **Maat**, analyzes VirusTotal scan reports to build **ML**-based labeling strategies. As seen in Figure 2, **Maat** starts by analyzing the VirusTotal scan reports of apps in the training dataset that are reanalyzed via VirusTotal and downloaded at different points in time (i.e., $t_0, t_1, \ldots, t_m$). The training dataset that **Maat** uses to train **ML**-based labeling strategies comprises the scan reports of apps in the *AMD+GPlay* dataset gathered between November 2018 and November 8th, 2019.

In phase (1) from Figure 2 we identify the VirusTotal scanners that achieve an average overall correctness score of at least 90% between November 2018 and November 8th, 2019 as the most correct scanners. **Maat** also finds the scanners that changed their verdicts at most 10% of the time (i.e., were stable 90% of the time), and deems them as the most stable scanners. The output of this phase is an intersection of the most correct and stable VirusTotal scanners. These scanners are considered in the next phase to extract features from the scan reports. The exact processes we adopted to find the most correct and stable scanners are detailed in Section 4.2 and Section 4.3, respectively.

In phase (2), we extract features from the VirusTotal scan reports of apps in the *AMD+GPlay* dataset. That is, each app in the dataset, will be represented as a vector of numerical features extracted from its most recent VirusTotal scan report. There are two types of features we extract from the reports, namely, *engineered* features and *naive* features.

Engineered features attempt to leverage the insights we gained from the previous phases (e.g., which scanners are correct and stable), and combine them with attributes retrieved from the VirusTotal scan reports of training apps that reflect domain knowledge of malware detection (i.e., attributes that help discern between malicious and benign apps). That is, we did not devise all features in the engineered feature set. As discussed later in Section 4.4, engineered features can be broken into three categories. Firstly, based on the output from phase (1), we consider the verdicts given to apps in the training dataset only by the set of most correct and stable scanners. Secondly, despite its dynamicity, VirusTotal scan reports are assumed to mature over time. That is, the older the app, the more accurate the scan results and metadata in its scan report are expected to be, which might help with correctly labeling the app. To accommodate the impact of time on

the maturity of an app's scan report, we also include the number of scanners deeming an app as malicious (i.e., *positives*), the total number of scanners that scanned an app (i.e., *total*), the age of a scan report in years, and the number of times an app has been submitted for (re)analysis (i.e., *times_submitted*), in this feature set. Lastly, to capture any patterns that Android (malicious) apps share in terms of functionalities and runtime behaviors, we extract from the `VirusTotal` scan reports the permissions that apps request in their `AndroidManifest.xml` files, and the tags given to them by `VirusTotal` (e.g., *checks-gps*, *contains-elf*, *sends-sms*). The exact list of engineered features can be found on **Maat**'s website.

Naive features do not consider the outputs of phase (1). With naive features, we consider **only** the verdicts given by *all* `VirusTotal` scanners to the apps in the training dataset. So, the feature vector extracted from a `VirusTotal` scan report will be a sequence of integers depicting the labels given by each scanner to an app (i.e., $-1$ for not scanned, 0 for scanned and deemed benign, and 1 for scanned and deemed malicious). For example, assume that the scan report of an arbitrary app ($\alpha^*$) contained scan results of three scanners, that respectively deemed ($\alpha^*$) as malicious, malicious, and benign; the feature vector depicting this scan report will be ($\hat{x}_{\alpha^*} = (1, 1, 0)$). With naive features, we allow **ML**-based labeling strategies to utilize the verdicts of all `VirusTotal` scanners regardless of their correctness or stability.

We envision the process of utilizing the features extracted from `VirusTotal` scan reports to label apps as a series or combination of questions, such as *how many scanners deem the app malicious? how old is the app? does a renowned scanner (e.g.,* `AVG`*) deem the app as malicious?* The ML classifier that mimics this model, we reckon, is a decision tree. In order not to rely on the decisions made by a single tree, **Maat** trains **ML**-based labeling strategy as a collection of trees or a *random forest* using the framework `scikit-learn`. To estimate the hyperparameters (e.g., the maximum depth each tree is allowed to grow), that train the most effective forests, we use the technique of *grid search* [33] to select from among a set of parameters listed in on **Maat**'s website. These parameters include *criterion*, *max_depth*, *max_features*, and *min_samples_split*.

Using the *max_features*, such decision trees can be instructed to consider a maximum number of randomly selected features each time it splits the tree into sub-tree. If the feature set contains noisy features, there is a chance that using *max_features* would lead to training random forests incapable of effectively classifying apps, despite comprising 100 decision trees. Phase (3) is an **optional** phase that selects the most informative features extracted from the training dataset's `VirusTotal` scan reports, so that decision trees that use the *max_depth* parameter randomly select a feature from a set of more informative ones. To avoid having to choose the number of features to select arbitrarily, we utilize the `SelectFromModel` [32] technique to select the most informative features automatically. In essence, this technique selects features based on the importance given to them by a model (e.g., logistic regression, support vector machines, decision trees). For example, during training, decision trees iteratively utilize a criterion (e.g., Gini index), to decide upon the next feature to consider in splitting data points into two, or more, classes; in our case, this feature could be a scanner's verdict regarding the label of an app. Ultimately, the trained tree will compile a set of features that it used during splitting and assign an *importance* value to each one of them. The `SelectFromModel` feature selection technique uses such importance values and returns the user those features with importance values more than a preset threshold (i.e., $1 \times 10^{-5}$ in the case of decision trees). For our experiments, we rely on decision trees as the model used by the `SelectFromModel` technique to extract the most informative features.

In phase (4), **Maat** uses the features extracted from the `VirusTotal` scan reports of training apps—and possibly further selected in phase (3)—to train a random forest. This forest that takes a vector of numerical features extracted from an app's `VirusTotal` scan report and returns a label depicting the class of the app (i.e., either 1.0 for malicious or 0.0 for benign). Effectively, this random

forest is an **ML**-based labeling strategy. In classifying the scan reports, we use the `predict` method; this method takes the probabilistic output of the `predict_proba` method that a sample belongs to each class and assigns the sample to the class with the highest probability. In phase (5), given the `VirusTotal` scan report of an arbitrary Android app, the report is represented as a feature vector that matches the features used by the random forest (e.g., naive versus engineered features), and is used to predict the app's class.

### 4.2 Correctness of `VirusTotal` Scanners

As part of extracting engineered features to train **ML**-based labeling strategies, in this section, we describe the process adopted by **Maat** to find the set of the most accurate `VirusTotal` scanners during a given period of time. Given the scan reports of apps in the training dataset gathered over a period of time, **Maat** relies on Mohaisen et al.'s definition of scanner *correctness* to identify the set of correct `VirusTotal` scanners over this period. In [19], Mohaisen and Alrawi defined correctness as follows: For a given dataset, the correctness of an antiviral scanner is the number of correct detections normalized by the size of the dataset. So, for each `VirusTotal` scanner, **Maat** calculates the number of apps in the *AMD+GPlay* dataset that it managed to correctly label as malicious, and divides that number by the size of the dataset. The correctness scores of each scanner at different points in time are then averaged. Scanners whose correctness scores are greater than or equal to 0.90 are included in the set of correct `VirusTotal` scanners. We chose the number 0.90 to tolerate any fluctuations in the correctness rate that may occur, for example, due to changing policies on how to label ambiguous malware types, such as `Adware`, or the temporary use of inadequate versions of scanners. So, the definition of a correct scanner we adopt in this article is

**Definition**

A correct `VirusTotal` scanner is one that is able to correctly label apps in a given dataset of pre-labeled apps, such that the number of correctly labeled apps over the total number of apps in the dataset is greater than or equal to 0.90.

Using this criterion, we retrieved a set of 18 scanners. This set of scanners retrieved by **Maat** from the *AMD+GPlay* dataset between November 2018 and November 8th, 2019 noticeably differs from the 10 scanners Arp et al. used in 2014 to label apps in their *Drebin* dataset [3]: only 5 out of 10 *Drebin* scanners are included in **Maat**'s list of correct scanners. The authors of *Drebin*, Arp et al., used a different set of Android apps and older versions of scan reports. This implies that the set of correct scanners might change from one set of Android apps to another and from time to time. In other words, it seems that there is no set of `VirusTotal` scanners that are universally correct on different datasets at different points in time.

To verify this, we retrieved the set of scanners that maintained correctness scores of 90% or higher between July 5th, 2019 and November 8th, 2019 for the *AMD*, *AMD+GPlay*, *GPlay*, *Hand-Labeled*, and *Hand-Labeled 2019* datasets (i.e., the datasets with known ground truths). As seen in Table 2, the set of `VirusTotal` scanners that have correctness rates of at least 90% differ from one dataset to another, despite being based on scan reports downloaded within the same period. Only a set of five scanners is shared by all datasets, viz., `ESET-NOD32`, `Fortinet`, `Ikarus`, `McAfee`, and `SymantecMobileInsight`. As for the *GPlay* dataset, a total of 58 correct scanners were found to be correct, which we could not fit in the table.

Moreover, we noticed that the larger the number of apps pre-labeled as benign in a dataset, the bigger the set of correct scanners we can retrieve from it. In theory, scanners should be assessed equally based on their ability to correctly label malicious and benign apps. However, mislabeling a benign app as malicious (i.e., false positives), is known to have more of a negative effect on the

Table 2. The Set of VirusTotal Scanners That Had Correctness Rates of at Least 90% between July 5th, 2019 and November 8th, 2019

| Dataset | Scanner(s) | Total |
|---|---|---|
| *AMD+GPlay* (49.06% benign + 50.04% malicious) | AhnLab-V3, Avira, CAT-QuickHeal, Comodo, Cyren, DrWeb, **ESET-NOD32**, F-Secure, **Fortinet**, **Ikarus**, K7GW, MAX, **McAfee**, NANO-Antivirus, Sophos, **SymantecMobileInsight**, TheHacker, Trustlook | 18 |
| *AMD* (malicious) | Avira, CAT-QuickHeal, DrWeb, **ESET-NOD32**, F-Secure, **Fortinet**, **Ikarus**, MAX, **McAfee**, NANO-Antivirus, Sophos, **SymantecMobileInsight** | 12 |
| *Hand-Labeled* (76% benign + 24% malicious) | AhnLab-V3, CAT-QuickHeal, Cyren, **ESET-NOD32**, **Fortinet**, **Ikarus**, K7GW, **McAfee**, Sophos, **SymantecMobileInsight**, Trustlook | 11 |
| *Hand-Labeled 2019* (90% benign + 10% malicious) | Ad-Aware, AegisLab, AhnLab-V3, Alibaba, Arcabit, Avast-Mobile, BitDefender, ClamAV, Cyren, DrWeb, **ESET-NOD32**, Emsisoft, F-Secure, FireEye, **Fortinet**, GData, **Ikarus**, Jiangmin, K7AntiVirus, K7GW, Kaspersky, Kingsoft, MAX, MalwareBytes, **McAfee**, McAfee-GW-Edition, MicroWorld-eScan, Microsoft, NANO-Antivirus, Qihoo-360, SUPERAntiSpyware, **SymantecMobileInsight**, Trustlook, ViRobot, Yandex, ZoneAlarm, Zoner | 37 |

Emboldened scanners depict the intersection of the sets of correct scanners of the four datasets.

reputation and, hence, the popularity of antiviral scanners than mislabeling a malicious app as benign [25]. Consequently, antiviral scanners are said to be reluctant to label apps as malicious and opt for assigning a label of benign to apps by default. This reluctance makes it difficult to assess whether a benign label given by an antiviral scanner to an app is a result of analyzing the apps and consciously labeling it as benign or just a default label. Labeling an app as malicious, however, may imply a thorough analysis process that led to deeming an app as such, especially given the negative impacts of false positives. So, we speculate that benign apps alone do not reveal the detection ability of an antiviral scanner. Unfortunately, we cannot assert why and how antiviral scanners deem apps as malicious or benign without access to their internal analysis and detection processes.

What we learned from this measurement is that the set of VirusTotal scanners that are correct over time might change depending on (a) the dataset itself and its composition in terms of benign and malicious apps, and (b) the time period within which the VirusTotal scan reports of those apps were gathered. However, we noticed that there is a subset of scanners that persist across different datasets, viz., the five scanners ESET-NOD32, Fortinet, Ikarus, McAfee, and SymantecMobileInsight. So, is there a universal set of VirusTotal scanners that are correct across different points in time and different datasets? The answer depends on the definition of a universal set. If researchers expect that set of scanners to maintain the same scanners and cardinality, then the answer is no. However, it is expected for a small set of VirusTotal scanners to persist within the set of correct scanners longer than others.

Now the question is why and how does time impact the performance of an antiviral scanner on VirusTotal? One possible answer is the scanners suffer technical difficulties, which is reflected in their detection performances. A more concerning possibility is that VirusTotal changes the set of scanners it includes in the scan reports of apps across time. Since we calculate correctness based on the verdicts of scanners found in VirusTotal's scan reports, excluding a scanner's verdict from such scan reports will give the wrong impression that the scanner was unable to classify the apps correctly and, in turn, decrease the scanner's correctness score. **Maat** supports tabulating and visualizing the correctness of VirusTotal scanners to provide its users with further insights about the performances of different scanners over time. For example, Figure 3 displays the correctness

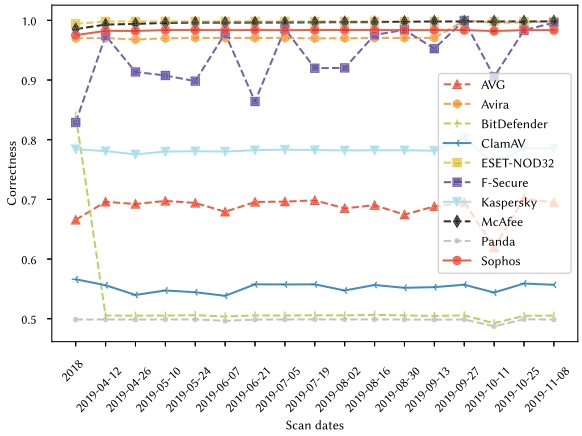



Fig. 3. The overall correctness rates of the *Drebin* scanners on apps in the *AMD+GPlay* dataset between November 2018 and November 8th, 2019.

scores of the *Drebin* scanners: AntiVir[8], AVG, BitDefender, ClamAV, ESET, F-Secure, Kaspersky, McAfee, Panda, and Sophos.

The correctness scores in the figure imply that some scanners are more correct and stable than others, which is expected. However, the performance of F-Secure and BitDefender raises the following concerns. The correctness of F-Secure appears to noticeably oscillate on a frequent basis. For example, the overall correctness of F-Secure steeply dropped from above 0.90 on June 7th, 2019 to around 0.70 on June 21st, 2019 only to increase to 0.90 2 weeks later on July 5th, 2019. While it might be acceptable that the performance of an antiviral scanner drops and remains low, such as the case of BitDefender, there is no apparent reason for the frequent fluctuations in performance. In what appears to be a random decision, we found that VirusTotal excludes the verdicts of F-Secure from the scan reports of apps in the *AMD+GPlay* dataset. For example, on the dates April 26th, 2019, May 10th, 2019, and May 24th, 2019, the verdicts of F-Secure were included in the scan reports of *all*, 98%, and 97% of apps in the dataset, respectively. Consequently, due to the exclusion of the scanner from the scan reports of some apps, the correctness scores of F-Secure on those dates were respectively 0.92, 0.90, and 0.88. This leads to VirusTotal having the following limitation:

**VirusTotal Limitation 2**

VirusTotal changes the set of scanners it includes in the scan reports of apps over time by including and excluding the verdicts of scanners regardless of the quality of those verdicts.

Second, despite its mediocre performance as reported by VirusTotal, BitDefender continues to be given good reviews by users on the Google Play marketplace and, more importantly, on platforms that assess the effectiveness of antiviral software such as *AV-Test* [11]. Given that VirusTotal states that the versions of scanners it uses "*may differ from commercial off-the-shelf products. The [antiviral software] company decides the particular settings with which the engine should run in VirusTotal*" [38], we compared the VirusTotal version of BitDefender against the one that can be found on the Google Play app marketplace at that time. We found that, as of

[8]AntiVir refers to the antiviral scanner developed by Avira, which is the name that VirusTotal uses for this scanner now.

September 2019, the version used by VirusTotal's for BitDefender is 7.2, whereas the versions available on Google Play have codes between 3.3 and 3.6. The 7.2 version of BitDefender corresponds to a free edition version developed for Windows-based malware that targets older versions of Windows such as Windows XP [16]. Given the positive reputation that BitDefender has in the market, we hypothesized that using its adequate version (i.e., the one that is designed to detect Android malware), would yield a detection performance better than the version on VirusTotal. So, we downloaded and installed the latest version of the BitDefender scanner from the Google Play marketplace, installed it on an **Android Virtual Device** (**AVD**), and used it to scan 10 apps randomly sampled from the *AMD* dataset. Unlike the results obtained from VirusTotal that the scanners are unable to detect any of those apps, we found that BitDefender detects 70% of the sampled apps as opposed to 20% using the 7.2 version.

**VirusTotal Limitation 3**

VirusTotal may replace the versions of scanners with inadequate ones that are not designed to detect Android malware presumably based on the request of the scanner's vendor or managing firm.

So, do some antiviral software companies believe that offering older versions on VirusTotal will encourage users to download their products instead of relying on the online version available on VirusTotal? Do such companies enforce a fee on VirusTotal in order to use their new products, which the latter did not agree to? Answering those questions is not in the scope of this article and, in fact, are impossible to answer on behalf of antiviral software companies.

### 4.3 Stability of VirusTotal Scanners and Scan Reports

Whether due to the dynamicity of VirusTotal or internal decisions, the examples of F-Secure and BitDefender in the previous section show that some VirusTotal scanners have unstable, fluctuating performance. To train reliable **ML**-based labeling strategies, **Maat** attempts to rely on VirusTotal scanners that do not often change the labels they assign to Android apps (i.e., stable scanners). **Maat** identifies stable VirusTotal scanners over a period of time by considering those scanners whose *certainty* score exceeds 0.90. A scanner's *certainty* score is calculated as follows. The labels given by the scanner to each app in the training dataset (i.e., *AMD+GPlay*), over a period of time are retrieved. The total number of the **most common** label (i.e., malicious versus benign) is divided by the total number of labels. For example, if the scanner ($\sigma$) had the labels $L = \{\texttt{malicious}, \texttt{benign}, \texttt{malicious}, \texttt{malicious}\}$ for an app ($\alpha$) over four points in time. Since the label malicious is the most common label in ($L$), the certainty score will be three (i.e., counts of malicious), divided by a total of four labels yielding a percentage of 0.75. We can represent this as the following formula $certainty(\sigma, \alpha) = \frac{count(common(L))}{size(L)}$. To calculate the certainty score for an entire dataset, **Maat** averages the scanner's certainty scores achieved on individual apps in this dataset. Similar to the case with the correctness scores, **Maat**'s default threshold of the certainty score is 0.90 to allow for marginal fluctuations in the labels given to apps by scanners, which might be a result of VirusTotal excluding those scanners from the apps' scan reports or changing their versions to inadequate ones. So, we define a stable VirusTotal scanner in this article as follows:

**Definition**

A stable VirusTotal scanner is one that achieves an average certainty score of at least 0.90 on apps in a given dataset over a period of time. This score indicates that, on average and regardless of the correctness of the assigned label, the scanner maintained the same label it assigns to an app 90% of the time.

Using this threshold, **Maat** retrieved a set of 44 scanners (i.e., 74.5% of all VirusTotal scanners), that had certainty scores of at least 0.90 on apps in the *AMD+GPlay* dataset between November 2018 and November 8th, 2019. Unfortunately, VirusTotal does not grant access to the history of scan reports of apps under academic licenses, which hinders our efforts to assess the stability of different scanners in periods before November 2018

**VirusTotal Limitation 4**

VirusTotal does not grant access to academic researchers to the history of scan reports of apps previously added and scanned on the platform, even if such apps were added by the academic community itself.

It is important to mention that any definition of stability or method to retrieve the most stable scanners, including this one, would suffer from two limitations. Firstly, there is no guarantee that the performance of a VirusTotal scanner will continue to be the same, either because of technical difficulties suffered by an antiviral software company or because of VirusTotal's utilization of different versions of a scanner, as seen in the case of BitDefender. Secondly, any definition of stability does not guarantee correctness. Over the past 3 years, we tracked the verdicts given by VirusTotal scanners to a repackaged, malicious version[9] of the K9 Mail open source app [1] that has been developed by one of our students during a practical course. Despite being a malicious app of type Ransom, the scanners continued to unanimously deem the app as benign since February 8th, 2017, even after analyzing and re-scanning the app. Another example is an app[10] that we repackaged 3 years ago; the app continued to be labeled as benign by all scanners until only K7GW recognized the app's malignancy in July 2019 and labeled it as a Trojan. As mentioned in the previous section, **Maat** is meant to help optimally utilize VirusTotal and interpret its scan results, but it cannot control either its dynamicity or correctness. So, since the set of stable scanners may change in the future as VirusTotal changes, it is recommended to retrain **Maat**'s **ML**-based labeling strategies that rely on the set of stable scanners (i.e., using engineered features) whenever possible. In Section 5 and Section 6, we discuss how often should **Maat**'s **ML**-based labeling strategies should be retrained.

### 4.4 Features Extracted from Scan Reports

In this section, we recap on the type of features that **Maat** extracts from the VirusTotal scan reports of its training dataset. As mentioned in Section 4.1, there are two types of features that **Maat** can extract from scan reports, namely, engineered and naive.

***Engineered Features.*** The engineered features **Maat** extracts from scan reports (listed online) can be divided into three categories. In the first category, **Maat** considers the verdicts of VirusTotal scanners that had correctness **and** certainty scores of at least 0.90. As discussed in Section 4.2, the performances of some correct scanners, such as F-Secure, fluctuate over time (i.e., not stable). Moreover, the stability of scanners, in general, does not guarantee correctness, as discussed in Section 4.3. To mitigate both issues, **Maat** relies on the verdicts given by the intersection of correct and stable scanners. The reason behind this is that the continuity of correctly labeling apps leads to the stability of labels. In other words, if a scanner is consistently accurate at giving the same, correct label to an app, it is ipso facto a stable scanner. This relationship between correctness and stability, as discussed before, does not go the other way around (i.e., stability does not imply correctness). Taking the intersection between the 18 correct scanners, the 44 stable scanners

[9] aa0d0f82c0a84b8dfc4ecda89a83f171cf675a9a.
[10] 66c16d79db25dc9d602617dae0485fa5ae6e54b2: A calculator app grafted with a logic-based trigger that deletes user contacts only if the result of the performed arithmetic operation is 50.

yields a set of 16 VirusTotal scanners that include all the correct scanners apart from F-Secure, which already exhibited fluctuation in detection performance (see Section 4.2), and Trustlook. The second category of features is based on attributes found in VirusTotal scan reports that imply the age and, perhaps, the maturity of the app and its scan report. **Maat** extracts the age of the app's scan report as the difference between the extraction date and that of *first_seen* along with the *times_submitted* attribute, the *positives* attribute, and the *total* attribute. In our analysis, we noticed that older malicious apps have higher ranges of *positives* and *total* values than newer ones. For example, as of November 8th, 2019, the malicious apps in the *AMD* dataset have an average *positives* value of 26.26±5.53: a number newly developed malicious apps and benign ones are unlikely to reach. So, we thought that such values might assist **Maat** to discern malicious, benign, and ambiguous malicious apps. The third category of engineered features is meant to approximate the structure and behavior of apps. Malicious apps tend to adopt similar structures, re-use libraries, exploit similar vulnerabilities, or share the same codebases. We assume that such trends can be reflected in the permissions these apps request and the tags assigned to them by VirusTotal. So, as part of the engineered features, we include the list of permissions (not) requested by the app and the tags (not) assigned to them by VirusTotal. In total, **Maat** extracts 372 features from the VirusTotal scan report of each app. As discussed in Section 4.1, users of **Maat** have the option to instruct the framework to select from among those 372 features more informative ones to train the **ML**-based labeling strategies.

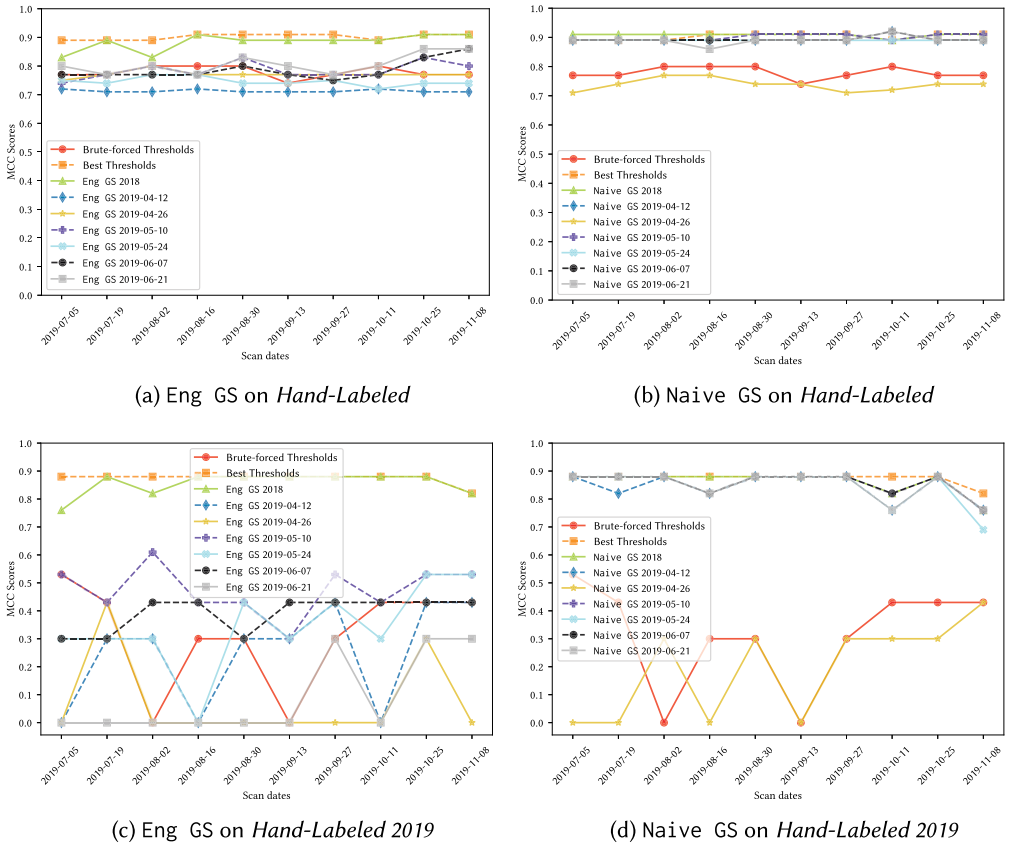
***Naive Features.*** Naive features comprise the verdicts of **all** VirusTotal scanners, regardless of their correctness or stability. With this set of features, as mentioned earlier, we allow **Maat**'s random forests to identify and choose the VirusTotal scanners that train the most effective **ML**-based labeling strategies. We use this type of features for three reasons. First, naive features are fast and easy to extract from VirusTotal scan reports. Second, we wish to investigate whether allowing **Maat**'s random forests to select scanners would yield a set of scanners that overlap with the ones identified using the measurements in Section 4.2 and Section 4.3. Third, we wish to assess whether the second and third categories of engineered features help **Maat** train better **ML**-based labeling strategies or hinder their performance. Without the further selection of features, the dimensionality of naive features is 60 scanners.

## 5 EVALUATING MAAT

As discussed in Section 1, the accurate labeling of previously analyzed apps is fundamental for the effectiveness of detection methods. So, the quality of labeling strategies can be assessed in terms of (a) their abilities to assign labels to previously analyzed apps (e.g., based on their VirusTotal scan reports) that accurately reflect their ground truths, and (b) the extent to which such accurate labeling contributes to the effectiveness of detection methods against apps that are yet to be analyzed and labeled. In this section, we evaluate the performance of threshold-based labeling strategies versus **Maat**'s **ML**-based labeling strategies according to the aforementioned two criteria in Section 5.1 and Section 5.2, respectively.

### 5.1 Accurately Labeling Apps

In this set of experiments, we trained **Maat**'s **ML**-based labeling strategies using VirusTotal scan report of apps in the *AMD+GPlay* re-analyzed and downloaded at different points in time between November 2018 and June 21st, 2019. Those labeling strategies are, then, used to label apps in the test datasets *Hand-Labeled* and *Hand-Labeled 2019* downloaded between July 5th, 2019 and November 8th, 2019. Effectively, we are testing whether those strategies trained at a given point in time can




(a) Eng GS on *Hand-Labeled*

(b) Naive GS on *Hand-Labeled*

(c) Eng GS on *Hand-Labeled 2019*

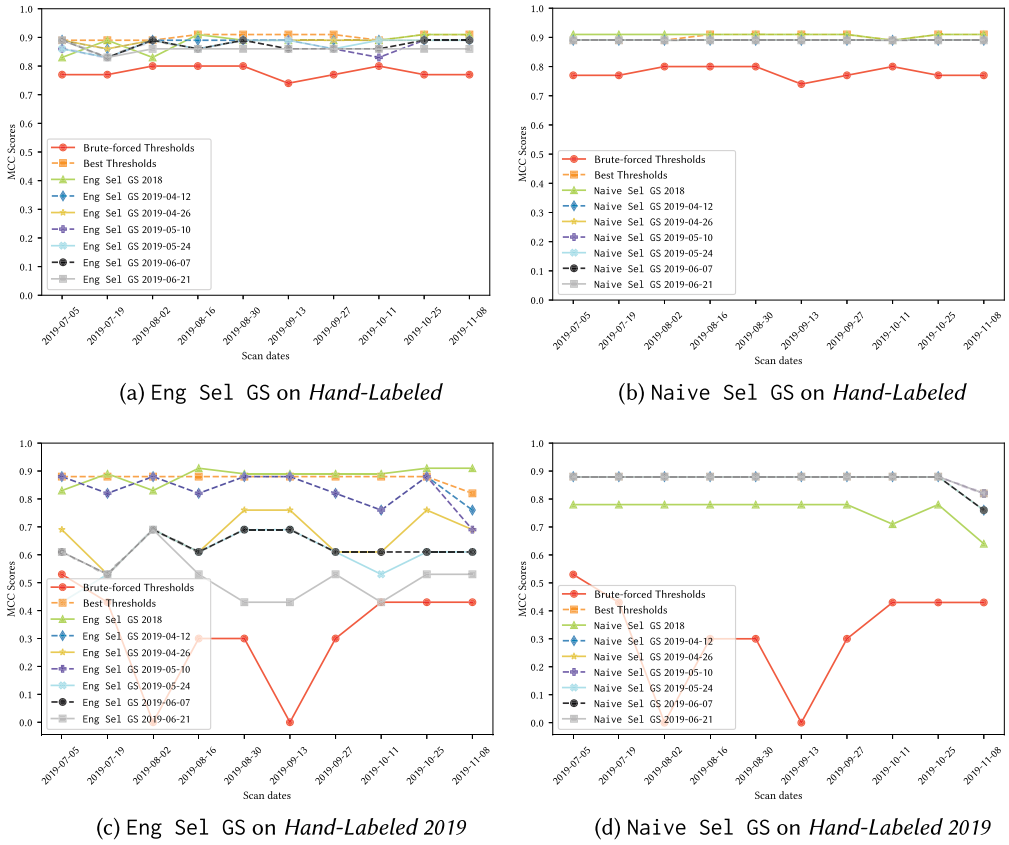
(d) Naive GS on *Hand-Labeled 2019*

Fig. 4. The labeling accuracies achieved by **Maat**'s **ML**-based labeling strategies Eng GS and Naive GS trained with scan reports downloaded between November 2018 and June 21st, 2019 against apps in the *Hand-Labeled* and *Hand-Labeled 2019* datasets over time.

maintain steady labeling accuracies in the future and, hence, do not need to be retrained as often as threshold-based strategies to accommodate VirusTotal's dynamicity.

In the experiments, we used **ML**-based labeling strategies that use the two types of features mentioned in Section 4.4 and trained using the technique of grid search. We also included labeling strategies that used the SelectFromModel technique to select the most informative features. For readability, we shorten the names of **ML** labeling strategies in the following manner: engineered and naive features are referred to as Eng and Naive, respectively, grid search is referred to as GS, and strategies that use selected features are referred to using Sel. For example, a labeling strategy that uses selected engineered features and the grid search technique will be referred to as Eng Sel GS.

In this section, we compare the labeling performance (in **MCC**) of the **ML**-based labeling strategies trained using scan reports from November 2018 against two types of threshold-based labeling strategies. For example, in Figure 4, the solid red line depicts the performance of threshold-based strategies using the optimal threshold at each scan date identified as follows. At each scan date, we calculated the labeling accuracy of each threshold between 1 and 60 against apps in the *AMD+GPlay* dataset. The threshold that yields the best **MCC** scores is considered as the optimal threshold for this scan date. For example, using scan reports of the *AMD+GPlay*




(a) Eng Sel GS on *Hand-Labeled* (b) Naive Sel GS on *Hand-Labeled*

(c) Eng Sel GS on *Hand-Labeled 2019* (d) Naive Sel GS on *Hand-Labeled 2019*

Fig. 5. The labeling accuracies achieved by **Maat**'s **ML**-based labeling strategies Eng Sel GS and Naive Sel GS trained with scan reports downloaded between November 2018 and June 21st, 2019 against apps in the *Hand-Labeled* and *Hand-Labeled 2019* datasets over time.

dataset re-scanned and downloaded on July 5th, 2019, the **brute-forced** optimal threshold was found to be 12 VirusTotal scanners (i.e., $vt \geq 12$), which is used to label apps in the *Hand-Labeled* and *Hand-Labeled 2019* datasets on July 5th, 2019. The other type of threshold-based labeling strategies we use (dashed orange line) depicts strategies that use the optimal threshold that would achieve the best **MCC** scores on each of the test datasets. These thresholds are found by trying different thresholds on the *Hand-Labeled* and *Hand-Labeled 2019* datasets. They are meant to simulate a scenario in which a researcher always manages to find the thresholds that yield the best possible **MCC** scores on the test datasets. Effectively, the performance of these thresholds depicts an upper bound on the performance of threshold-based labeling strategies on the test datasets.

The performance of the brute-forced thresholds in Figure 4 and Figure 5 demonstrates the main limitation of the brute-forcing approach to finding the optimal threshold of VirusTotal scanners at a given point in time: in order for the thresholds obtained by brute-forcing all possible values to generalize to new Android malware, the dataset used to find these optimal thresholds needs to be temporally diverse. Given that the *AMD+GPlay* dataset does not contain apps developed in 2019, the identified thresholds were too high (i.e., between 10 and 13 scanners) to match the low number of VirusTotal scans that deem the malicious apps in the *Hand-Labeled 2019* dataset as such. As for **Maat**'s **ML**-based labeling strategies, with a few exceptions, we found that (a) their **MCC** scores

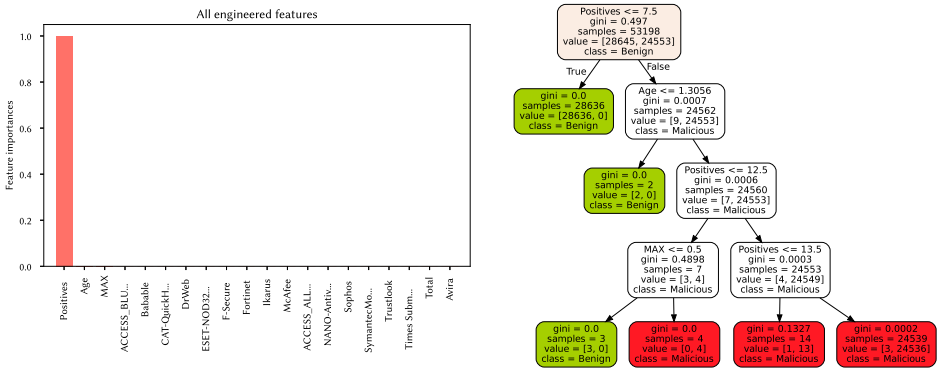
mimicked that of the best possible threshold at each scan date and (b) had stable **MCC** scores over the period of time between July 5th, 2019 and November 8th, 2019. Recall that for all scan dates, we used **ML**-based labeling strategies trained using `VirusTotal` scan reports that date back to different months in November 2018 and using a dataset of older apps (i.e., *AMD+GPlay*). This means that using `VirusTotal` scan reports of older apps, such labeling strategies can maintain decent, stable labeling accuracies **for at least a year** even against newer apps.

We made the following observations based on the **MCC** scores in Figure 4 and Figure 5:

- With a few exceptions, the performance of labeling strategies using engineered features (i.e., `Eng GS` and `Eng Sel GS`), significantly deteriorates on the *Hand-Labeled 2019* dataset. However, labeling strategies trained with `Eng Sel GS` seem to have higher scores than their counterparts trained with `Eng GS`, which is more evident against the *Hand-Labeled 2019* dataset, as seen in Figure 4(c) versus Figure 5(c).
- The **MCC** scores of labeling strategies using the naive features appear to be more stable than their counterparts using engineered features on both test datasets *Hand-Labeled* and *Hand-Labeled 2019*. Similar to the case with engineered features, the selected naive features `Naive Sel GS` enable the **ML**-based labeling strategies to achieve higher **MCC** scores and maintain almost the same performance over time regardless of when the labeling strategy was trained.
- The performance of both types of features seems to fluctuate over time, albeit not with the same rate or severity. Using selected features seems to decrease or remove such fluctuations, especially upon using the naive feature set.

*5.1.1 Features Learned by ML-Based Labeling Strategies.* In this section, we attempt to understand the labeling performances of different **Maat ML**-based labeling strategies by studying the structure of the random forests they comprise. For each type of features, we retrieve the list of important features learned by the aforementioned random forests during training using the *AMD+GPlay* dataset and briefly discuss the impact of such features on the labeling performance of their corresponding random forest. To further aid the reader with visualizing decision trees in such random forests, we display decision trees retrieved randomly from the random forest models trained using **Maat** (e.g., `clf.estimators_[random_idx]`). With the plotted decision trees, we do not assert that all trees within a given random forest have the same structure; we use them only for illustration purposes.

***Engineered Features.*** In Figure 6, one can notice that random forests trained using all engineered features primarily rely on the number of scanners that deemed an app as malicious (i.e., *positives*), to classify apps. The importance of this feature eclipses those of other ones with around 0.99 out of 1.0 importance. Figure 6(b) depicts an example of what decision trees in such random forests look like. Just by checking whether the *positives* features is less than seven scanners, the decision tree can correctly classify all but nine benign apps in the training dataset. Effectively, this type of features and the resulting decision trees implement a threshold-based labeling strategy of threshold seven (i.e., $\texttt{vt} \geq 7$). Similar to other threshold-based labeling strategies, this approach manages to effectively discern malicious apps that are old enough to have `VirusTotal` scan reports that reflect more consensus among scanner vis-à-vis their malignancy. Nevertheless, it cannot replicate the same performance against newer malicious apps, such as those in the *Hand-Labeled 2019* dataset, with immature scan reports and fewer scanners deeming them as malicious. In addition to that, `VirusTotal`'s regular manipulation of scanners included in apps' scan reports leads to fluctuations in the *positives* features relied on by such random forests,




(a) The importance of the top 20 engineered features learned by **Maat**'s random forests.

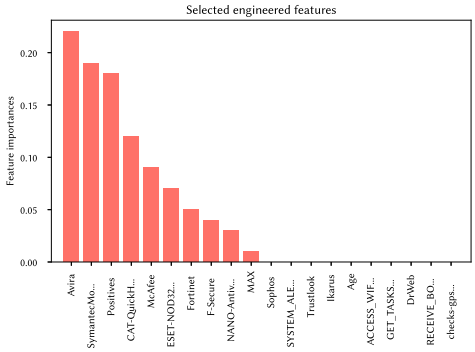
(b) Example decision tree

Fig. 6. The top 20 important features learned by **Maat**'s random forests using *all* engineered features and an example of what decision trees in those forests might look like.

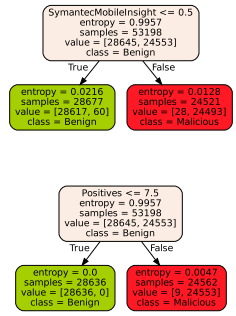
which might lead to incorrectly classifying malicious apps as benign because the *positives* values in their scan reports do not exceed the required threshold of seven scanners, as apparent in Figure 4(c).

***Engineered Selected Features.*** Allowing **Maat** to narrow down the number of engineered features before training the random forests has two impacts on the resulting decision trees, as seen in Figure 7. Firstly, the depth of trees decreases to one. More importantly, the features used by the decision trees in the random forest are more diverse. In addition to the *positives* attribute, the verdicts of a number of scanners—most of which belong to the set of correct scanners found in Section 4.2—are used. The labeling accuracies of random forests trained with selected engineered features are higher than the accuracies of their counterparts trained with all engineered features. However, against apps in the *Hand-Labeled 2019* dataset, a similar fluctuation of **MCC** scores can be noticed. We speculate that relying on the *positives* attribute might be the reason behind this fluctuation. In other words, if we only consider the verdicts of `VirusTotal` scanners as features to train **Maat**'s random forests (i.e., naive features), the aforementioned fluctuations may smoothen out.

***Naive Features.*** As expected and seen in Figure 8(a), the features used by **Maat** to train random forests are the verdicts of a subset of the scanners that are typically found in a `VirusTotal` scan report. This subset contains a mixture of scanners we found to be correct over time against apps in the *AMD+GPlay* dataset and others that do not belong to this set, which can be noticed in the example decision tree in Figure 8(b). We made the following observations about the structures of the decision trees in these forests. First, we found that the depth of the decision trees ranges from three to four. One possible reason behind this is that the verdicts of some scanners might be absent from the `VirusTotal` scan reports of some apps courtesy of `VirusTotal`'s second and third limitations. So, to complement the missing verdicts of one scanner, a decision tree checks those of other scanners. Second, we noticed that the structure of the decision trees in these random forests seems to be ruling out malicious apps at each split until only, or mostly, benign apps remain. For example, after checking the verdict of the `MAX` classifier, 24,075 (≈98%) of the malicious apps could already be classified. The remaining 494 apps are classified after checking the verdicts of other

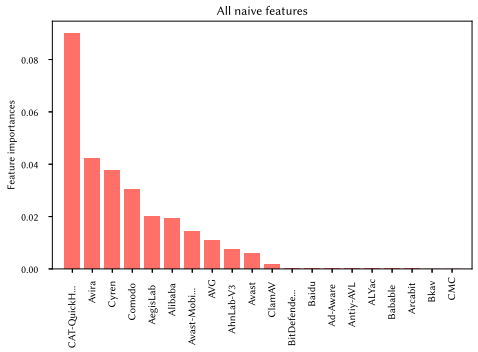



(a) The importance of the top 20 selected engineered features learned by **Maat**'s random forests.

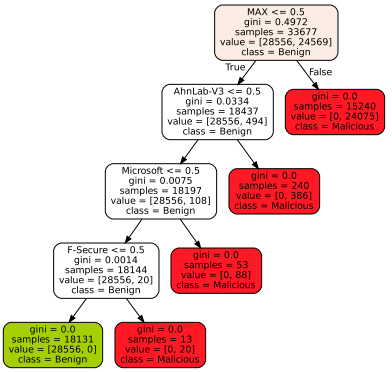



(b) Example decision trees

Fig. 7. The top 20 important features learned by **Maat**'s random forests using *selected* engineered features (i.e., applying phase (3) of **Maat**), and two examples of how decision trees in those forests might look like.




(a) The importance of the top 20 naive features learned by **Maat**'s random forests.




(b) Example decision tree

Fig. 8. The top 20 important features learned by **Maat**'s random forests using *all* naive features and an example of what decision trees in those forests might look like.

scanners until only 22 apps remain, which are misclassified as benign. The **MCC** scores of this type of random forests are higher than the scores of their counterparts trained using engineered features and fluctuate less on the *Hand-Labeled 2019*. This behavior tends to support the presumption that removing features, such as *positives*, *age*, and *times_submitted*, from the feature set helps **Maat** to train better performing random forests.

***Naive Selected Features.*** After feature selection, **Maat** only uses the verdicts of 16 scanners, seen in Figure 9, to train the random forests. The majority of such scanners were among the set of correct scanners retrieved by **Maat**. The structures of the corresponding decision trees are similar to those in forests trained using all naive features: malicious apps are classified first after checking the verdicts of `VirusTotal` scanners until only benign apps remain. However, it seems that relying on the verdicts of `VirusTotal` scanners that are correct and stable (see Section 4.2 and Section 4.3) during a given time period allows the random forests to (a) achieve higher **MCC** scores than those

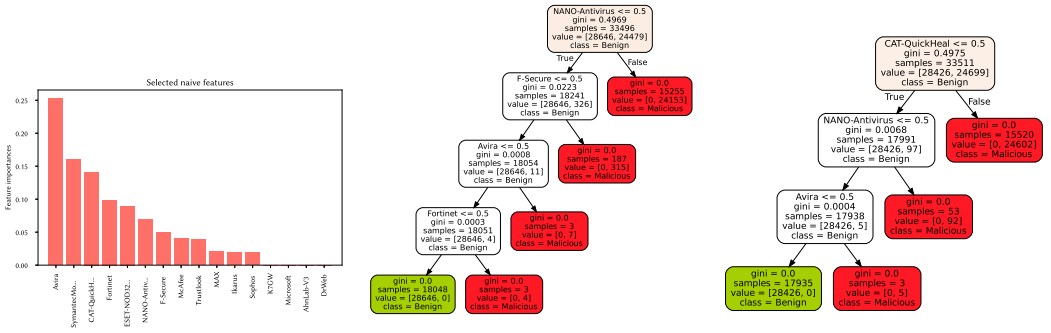



Fig. 9. The importance of all 16 features learned by **Maat**'s random forests using *selected* naive features (i.e., applying phase (3) of **Maat**), and two examples of what decision trees in those forests might look like.

achieved using other feature types, and (b) exhibit fewer and less severe fluctuations over time on both test datasets.

In conclusion, we gained the following insights:

- We found that naive features train **ML**-based labeling strategies that have higher and more stable **MCC** scores on both test datasets than their counterparts trained using engineered features.
- For both types of features, we found that selecting the most informative features using `SelectFromModel` before training boosts the labeling accuracies of **Maat**'s **ML**-based labeling strategies and relatively smoothens the fluctuations of their **MCC** scores over time that result from `VirusTotal`'s manipulation of scan reports.
- The structure of **Maat**'s **ML**-based labeling strategies using selected naive features appears to be more resilient to `VirusTotal`'s dynamicity than threshold-based labeling strategies in general. First, recall that `VirusTotal` sometimes uses inadequate versions of otherwise competent scanners (i.e., third limitation), which causes the verdicts of such scanners to be incorrect, especially against malicious apps. By selecting only the verdicts of `VirusTotal` scanners that are correct, **Maat**'s **ML**-based labeling strategies can mitigate this limitation. Second, basing the label of an app on the verdicts of between three and four correct `VirusTotal` scanners helps these **ML**-based labeling strategies yield accurate labels even if the verdicts of one or two of such scanners were not present in the scan report of an app (i.e., `VirusTotal`'s first limitation). Furthermore, since the final label of an app is based on the labels given by 100 decision trees using different combinations of `VirusTotal` scanners, the likelihood of this aspect of `VirusTotal`'s dynamicity to impact the decision of enough trees to lead to an incorrect final decision is low.

*5.1.2 Sensitivity to* `VirusTotal`*'s Dynamicity.* In this section, we attempt to find out how `VirusTotal`'s dynamicity impacts the structure and labeling accuracy of **Maat**'s **ML**-based labeling strategies.

***Impact of* `VirusTotal`*'s Dynamicity During Training.*** In Section 4.1, we mentioned that **Maat** uses the technique of grid search to find the hyperparameters that yield the most accurate random forests that constitute **Maat**'s **ML**-based labeling strategies. One of those hyperparameters controls the maximum allowed depth of all decision trees in the random forest (i.e., *max_depth*). We varied the value of *max_depth* to be 1, 4, 10, and `None` (i.e., no limitation), and allowed the technique of grid search to identify the value that yields the best validation accuracy

($\frac{TP+TN}{P+N}$) achieved using the technique of cross-validation, which we set to be 10-fold. In our case, this accuracy is calculated by splitting the scan reports of apps in the *AMD+GPlay* dataset into 10 folds, train the random forests using 9 of those, and using the remaining one-tenth as a validation dataset. The final validation accuracy is an average of all 10 accuracies achieved on the 10 validation datasets. We found that the validation accuracies achieved by random forests of different depths are very close to one another. For example, for `Eng Sel GS` **ML**-based labeling strategies, we found that the validation accuracies achieved by random forests with *max_depth* values of 1, 4, 10, and `None` were 1.0, 0.9999794703346335, 0.9999794703346335, and 0.9999589490951507, respectively. Based on those values and despite the insignificant difference in validation accuracies, the grid search algorithm suggested the random forests with *max_depth* of 1 as the best random forest. Using the *AMD+GPlay* dataset during training, an insignificant difference in validation accuracy of 0.00002053 results from an average misclassification of 0.1 (i.e., $4{,}871 \times 0.00002053 = 0.100011895$) apps in the validation datasets of size 4,871 apps (i.e., $48{,}715 \times 0.1 = 4{,}871$). In other words, it takes a few apps to be misclassified in one fold during validation to force **Maat** to choose one value for *max_depth* over another. The role that `VirusTotal`'s dynamicity plays in this case is that the manipulation of a scan report of a few apps in the training dataset (e.g., removing the verdicts of a number of scanners), at one point in time might result in enough misclassifications during validation that leads to choosing a value for *max_depth* that does not yield the best labeling accuracy.

***Impact of `VirusTotal`'s Dynamicity During Test.*** Another aspect of the **ML**-based labeling strategies' sensitivity to `VirusTotal`'s dynamicity is evident in the fluctuations of their **MCC** scores over time, especially on the *Hand-Labeled 2019* dataset (i.e., during testing the labeling accuracy of the trained strategies). The frequent change in the scanners used in the scan reports of these new apps affects the attributes extracted from these scan reports (e.g., *positives*), and the verdicts given by scanners. In turn, the performance of labeling strategies that use engineering and naive features is affected. As discussed earlier, using selected features seems to stabilize the performance of **ML**-based labeling strategies, especially those using naive features. However, on November 8th, 2019, one can notice the decrease in the **MCC** scores of even the most stable strategies (i.e., `Naive Sel GS`). In Section 4.2, we discussed and demonstrated `VirusTotal`'s manipulation of verdicts in scan reports, which particularly impacts the correct classification of new malicious apps (e.g., in the *Hand-Labeled 2019* dataset). Since `Naive GS` and `Naive Sel GS` labeling strategies solely rely on the verdicts of `VirusTotal` scanners to label apps as malicious, removing a number of them from the scan reports of apps is expected to impact the decision trees that rely on their verdicts to label apps. For example, given that they are among the correct and stable scanners that `Naive Sel GS` relies on, removing the verdicts of the `ESET-NOD32`, `Fortinet`, and `Ikarus` scanners from the scan report of the malicious app `90e6ac481fdd497f152234f1cd5bec6d40f50037` will have a negative impact on decision trees that employ these scanners. Fortunately, the decision trees of such labeling strategies combine the verdicts of different scanners, which reduces misclassifications. That is, the highest drop in **MCC** scores for the `Naive Sel GS` strategies between October 25th, 2019 and November 8th, 2019 is 0.14, which is a decrease of 18%.

### 5.2 Enhancing Detection Methods

In Section 1, we discussed that inaccurate labeling might have a negative impact on the detection performance of otherwise effective detection methods. By addressing this issue of labeling accuracy, we can contribute to helping the research community focus on developing effective detection methods rather than being consumed by devising labeling strategies that accurately label apps in their training datasets. So, the second criterion we use to evaluate the applicability

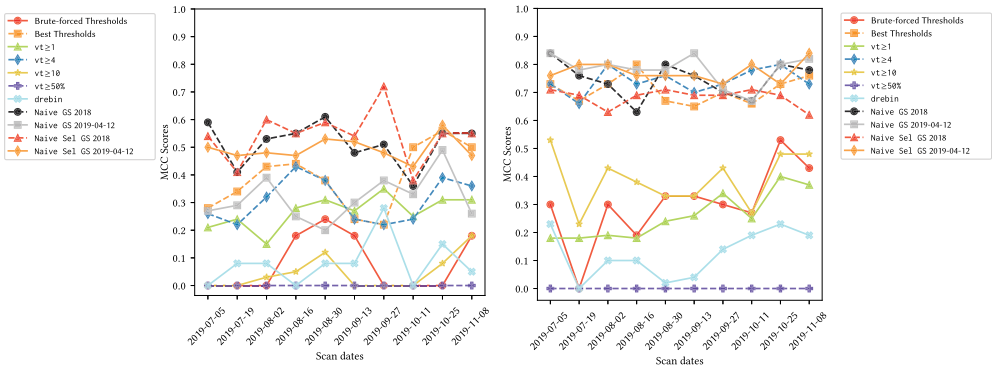
of **Maat**'s **ML**-based labeling strategies is their ability to contribute to training more effective **ML**-based detection methods. We focus on **ML**-based detection methods given their popularity within the academic community [4, 22, 36, 37, 43].

Typically, developers of such methods would use a dataset of feature vectors extracted from apps and pre-labeled as, for example, malicious and benign. This dataset is used as both a training and a test dataset using *K-Fold* cross-validation to get an objective measure of the detection method's classification performance. Unfortunately, the only large dataset of labeled apps we possess is that of *AMD+GPlay*, which does not contain new apps with immature `VirusTotal` scan reports that might challenge **Maat**'s **ML**-based labeling strategies, as demonstrated in the previous section. Consequently, we deferred to the classic setting of evaluating **ML**-based detection methods, viz., training a model using a dataset and testing it using another one whose apps were not used during training. In particular, we train different **ML** classifiers using static features extracted from the *AndroZoo* dataset, which depict information about the apps' components [30, 31], the permissions they request [3, 42], the API calls found in their code bases [26, 44], and the compilers used to compile them [35]. The feature vectors are then labeled using different threshold-based and **ML**-based labeling strategies. We test the detection abilities of such classifiers by assessing their ability to label apps in the *Hand-Labeled* and *Hand-Labeled 2019* datasets correctly. For readability, we use the shorter term *classifier was labeled* instead of *classifier whose feature vectors were labeled*.

We utilize a detection method that is renowned in the research community and has been used by different researchers as a benchmark [7], namely, *Drebin* [3]. The *Drebin* approach comprises three components: a linear support vector machine *LinearSVC* to classify apps, a set of static features extracted from Android apps that spans all app components, permissions, **Uniform Resource Locators** (**URLs**), and so forth, and the `drebin` labeling strategy. In particular, given a dataset of Android **APK** archives, the *Drebin* method collects all component names (e.g., activities), permissions, and **Uniform Resource Locator** (**URL**)s found in all of those apps, and considers each unique occurrence as a feature. For each identified feature, if an app's **APK** archive contains this feature (e.g., requests a particular permission), the corresponding feature in the app's feature vector will set as 1.0; otherwise, it will be set as 0.0. Using an implementation of *Drebin*'s feature extraction algorithm, we extracted a total of 71,260 features from apps in the *AndroZoo*, *Hand-Labeled*, and *Hand-Labeled 2019* datasets. Those feature vectors are labeled based on the `VirusTotal` scan reports of their apps using the `drebin` labeling strategy we introduced in Section 1 and Section 4.2. Lastly, a linear **Support Vector Machine** (**SVM**) model is trained using the training feature vectors (i.e., *AndroZoo*).

In addition to *Drebin*, we use the following classifiers: ***K-Nearest Neighbors (KNNs)*** [29], ***Random Forest (RF)*** [30], ***Support Vector Machine (SVM)*** [3], and ***Gaussian Naive Bayes (GNB)***[11] [30]. Given that the **K-Nearest Neighbors** (**KNN**) and **Random Forest** (**RF**) classifiers can have different values for their hyperparameters, we used the technique of grid search to identify the classifiers that yielded the best validation accuracies and used them as representatives of those classifiers. We also varied the values of the primary hyperparameters of the **KNN** and **RF** classifiers as follows: we varied the number of nearest neighbors to consider by the **KNN** classifier to $K = \{11, 26, 51, 101\}$ and the number of decisions trees in the random forest to $\{25, 50, 75, 100\}$. Consequently, we refer to these classifiers as **KNN** and **RF**. To train these classifiers, we statically extracted numerical features from the **APK** archives of apps in the *AndroZoo*, which depict infor-

[11]Gaussian naive Bayes classifiers assume that features have a Gaussian distribution. We argue that the static features we extract from the apps indeed follow such distribution. For example, while a small number of apps request either a very small or a considerable amount of requests, the majority of apps request an adequate amount (e.g., in the lower tens).




(a) *Hand-Labeled* (b) *Hand-Labeled 2019*

Fig. 10. The **MCC** scores achieved by the *Drebin* classifiers labeled using different threshold-/**ML**-based labeling strategies against the *Hand-Labeled* and *Hand-Labeled 2019* datasets between July 5th, 2019 and November 8th, 2019.

mation about the apps' components [30, 31], the permissions they request [3, 42], the API calls found in their code bases [26, 44], and the compilers used to compile them [35].

The labeling strategies we consider in this experiment are the conventional threshold-based strategies of $\texttt{vt} \geq 1$, $\texttt{vt} \geq 4$, $\texttt{vt} \geq 10$, $\texttt{vt} \geq 50\%$, and `drebin`, the threshold-based labeling strategies that use thresholds brute-forced at each point in time, threshold-based labeling strategies that use the best threshold at each point in time, and **Maat**'s **ML**-based labeling strategies that had the highest **MCC** scores in the previous set of experiments, namely, `Naive GS` and `Naive Sel GS`. In assessing the performance of different **ML**-based detection methods, we are **not** concerned with absolute values that indicate the quality of the classifier and the features it is trained with. Instead, we are interested in their performance with respect to one another. Given that the main difference between such classifiers is that their training feature vectors were labeled using different labeling strategies, this experiment is concerned with verifying the hypothesis of *more accurate labeling leads to better detection.*

In Figure 10, we plot the classification performance of different *Drebin* classifiers against apps in the *Hand-Labeled* and *Hand-Labeled 2019* datasets in terms of **MCC** score. The feature vectors used to train each of those classifiers have been labeled using a different labeling strategy. Given that the performances of different classifiers are intertwined, we tabulate their **MCC** scores in Table 3 for better readability. Studying these performances, we made the following observations:

— The performance of *Drebin* classifiers is better on the *Hand-Labeled 2019* dataset than on the older *Hand-Labeled* dataset. The reason behind this is that apps in the *AndroZoo* dataset were developed between 2018 and 2019. Probably, features extracted from these apps are expected to be similar to apps in the *Hand-Labeled 2019* dataset more than to those in the *Hand-Labeled* dataset, especially since the *Drebin* feature set is designed to identify similarity in the components and features utilized by the Android apps in the training dataset. This proximity in feature space helps *Drebin*'s **Linear Support Vector Machine** (**SVC**) classifier identify patterns shared by malicious and benign apps better.

— We found that using best threshold at each point on time along with **Maat**'s **ML**-based labeling strategies helps the *Drebin* classifier achieve better **MCC** scores than threshold-based labeling strategies that use fixed thresholds over time (e.g., $\texttt{vt} \geq 1$, $\texttt{vt} \geq 10$, and $\texttt{vt} \geq 50\%$), and better than the Brute-forced Thresholds. Moreover, the average **MCC** scores of *Drebin*

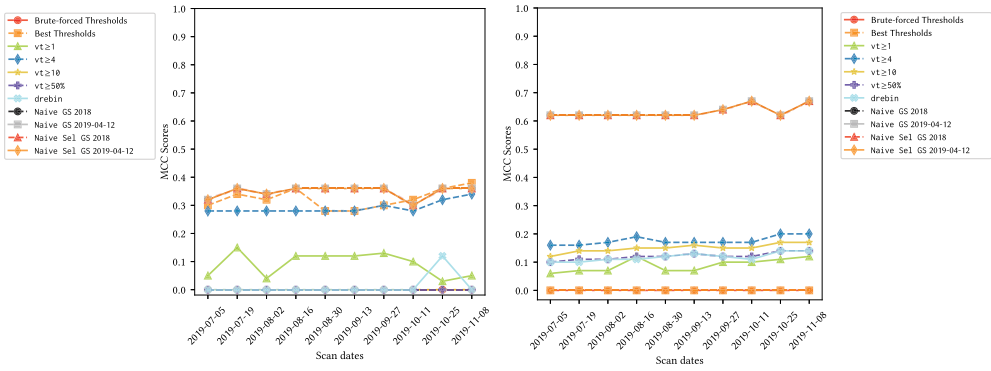
Table 3. Detailed View of the **MCC** Scores Achieved by the *Drebin* Classifiers Labeled Using Different Threshold-/**ML**-Based Labeling Strategies Against the *Hand-Labeled* and *Hand-Labeled 2019* Dataset between July 5th, 2019 and November 8th, 2019

| Labeling Strategy \ Scan Date | 2019-07-05 | 2019-07-19 | 2019-08-02 | 2019-08-16 | 2019-08-30 | 2019-09-13 | 2019-09-27 | 2019-10-11 | 2019-10-25 | 2019-11-08 |
|---|---|---|---|---|---|---|---|---|---|---|
| ***Hand-Labeled*** | | | | | | | | | | |
| Brute-forced Thresholds | 0.00 | 0.00 | 0.00 | 0.18 | 0.24 | 0.18 | 0.00 | 0.00 | 0.00 | 0.18 |
| Best Thresholds | 0.28 | 0.34 | 0.43 | 0.44 | 0.38 | 0.24 | 0.22 | 0.50 | 0.56 | 0.50 |
| vt ≥ 1 | 0.21 | 0.24 | 0.15 | 0.28 | 0.31 | 0.27 | 0.35 | 0.25 | 0.31 | 0.31 |
| vt ≥ 4 | 0.26 | 0.22 | 0.32 | 0.43 | 0.38 | 0.24 | 0.22 | 0.24 | 0.39 | 0.36 |
| vt ≥ 10 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| vt ≥ 50% | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| drebin | 0.00 | 0.08 | 0.08 | 0.00 | 0.08 | 0.08 | 0.28 | 0.00 | 0.15 | 0.05 |
| Naive GS 2018 | 0.59 | 0.41 | 0.53 | 0.55 | 0.61 | 0.48 | 0.51 | 0.36 | 0.55 | 0.55 |
| Naive GS 2019-04-12 | 0.27 | 0.29 | 0.39 | 0.25 | 0.20 | 0.30 | 0.38 | 0.33 | 0.49 | 0.26 |
| Naive Sel GS 2018 | 0.54 | 0.51 | 0.60 | 0.55 | 0.59 | 0.54 | 0.72 | 0.38 | 0.55 | 0.55 |
| Naive Sel GS 2019-04-12 | 0.50 | 0.47 | 0.48 | 0.47 | 0.53 | 0.52 | 0.48 | 0.43 | 0.58 | 0.47 |
| ***Hand-Labeled 2019*** | | | | | | | | | | |
| Brute-forced Thresholds | 0.30 | 0.00 | 0.30 | 0.19 | 0.33 | 0.33 | 0.30 | 0.27 | 0.53 | 0.43 |
| Best Thresholds | 0.73 | 0.67 | 0.73 | 0.80 | 0.67 | 0.65 | 0.70 | 0.66 | 0.73 | 0.76 |
| vt ≥ 1 | 0.18 | 0.18 | 0.19 | 0.18 | 0.24 | 0.26 | 0.34 | 0.25 | 0.40 | 0.37 |
| vt ≥ 4 | 0.73 | 0.66 | 0.80 | 0.73 | 0.76 | 0.70 | 0.73 | 0.78 | 0.80 | 0.73 |
| vt ≥ 10 | 0.53 | 0.23 | 0.43 | 0.38 | 0.33 | 0.33 | 0.43 | 0.27 | 0.48 | 0.48 |
| vt ≥ 50% | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |
| drebin | 0.23 | 0.00 | 0.10 | 0.10 | 0.02 | 0.04 | 0.14 | 0.19 | 0.23 | 0.19 |
| Naive GS 2018 | 0.84 | 0.76 | 0.73 | 0.63 | 0.80 | 0.76 | 0.70 | 0.67 | 0.80 | 0.78 |
| Naive GS 2019-04-12 | 0.84 | 0.78 | 0.80 | 0.78 | 0.78 | 0.84 | 0.71 | 0.67 | 0.80 | 0.82 |
| Naive Sel GS 2018 | 0.71 | 0.69 | 0.63 | 0.69 | 0.71 | 0.69 | 0.69 | 0.71 | 0.69 | 0.62 |
| Naive Sel GS 2019-04-12 | 0.76 | 0.80 | 0.80 | 0.76 | 0.76 | 0.76 | 0.73 | 0.80 | 0.73 | 0.84 |

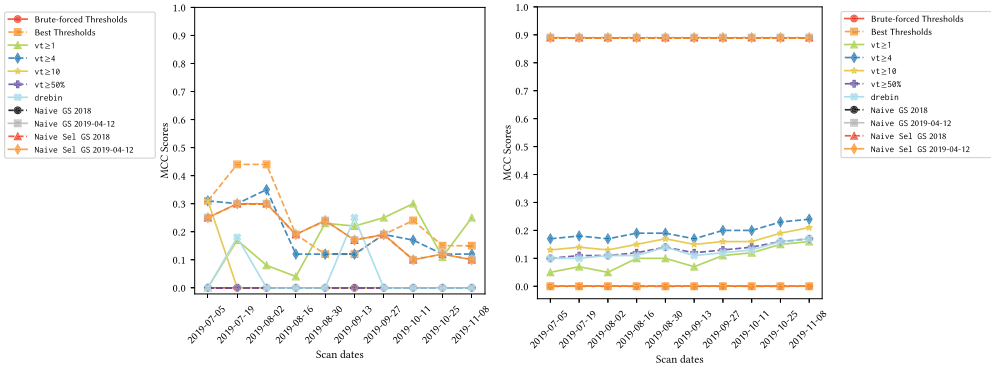
classifiers labeled using **Maat**'s **ML**-based labeling strategies is higher than that of classifiers labeled using the Best Thresholds at each scan date. As seen in Section 5.1, **Maat**'s **ML**-based labeling strategies using (selected) naive features and the Best Thresholds labeling strategies achieved the highest and steadiest **MCC** scores that depict their labeling accuracies.

— The results in Table 3 show that using labeling strategies that had the highest **MCC** scores for labeling accuracy contributes to training **ML**-based detection methods that achieve the best **MCC** scores, which supports the hypothesis that the more accurate and reliable a labeling strategy is, the more effective are the detection methods that rely on apps labeled by such a strategy. However, we noticed a number of exceptions that contradict this hypothesis. For example, the `Naive GS` **ML**-based labeling strategies trained in November 2018 and the `Naive GS` strategies trained on April 12th, 2019 had almost the same labeling accuracy against apps as the *Hand-Labeled* database. Despite this similarity, the *Drebin* classifiers labeled using the latter strategy noticeably underperforms in comparison to the former one. This behavior switches on the *Hand-Labeled 2019* dataset: the performance of *Drebin* classifiers labeled using `Naive GS` 2018 almost mimics that of those labeled using `Naive GS` trained on April 12th, 2019, despite the former being less able to label apps in this dataset accurately.

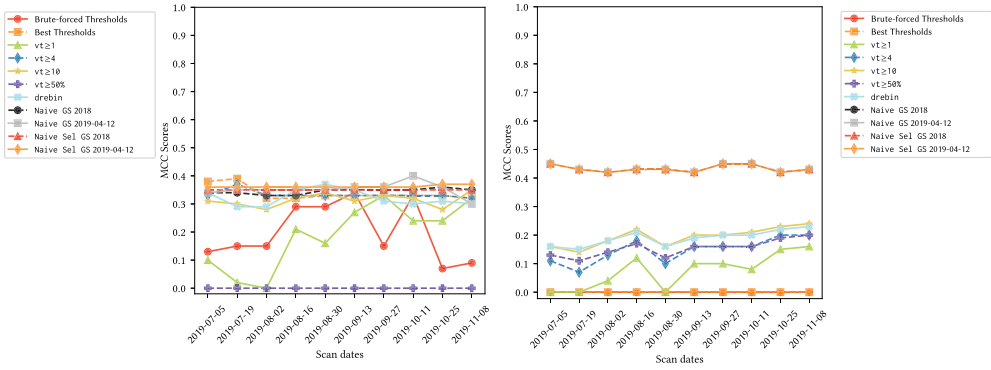
In general, we noticed that the **MCC** score of each *Drebin* classifier differs depending on the utilized labeling strategy and the scan date of the `VirusTotal` scan reports this strategy uses to label apps. In addition to these two factors, we noticed that the utilized classifier also impacts these scores. Figure 11 shows the **MCC** scores achieved by the **KNN**, **RF**, and **GNB** classifiers against the same test datasets using the same labeling strategies. Similar to the *Drebin* classifier, the **MCC** scores achieved by these three classifiers show that **Maat**'s **ML**-based labeling strategies contribute to training more effective detection methods than their threshold-based counterparts,




(a) KNN with *Hand-Labeled* (b) KNN with *Hand-Labeled 2019*




(c) RF with *Hand-Labeled* (d) RF with *Hand-Labeled 2019*




(e) GNB with *Hand-Labeled* (f) GNB with *Hand-Labeled 2019*

Fig. 11. The **MCC** scores achieved by the **KNN**, **RF**, and **Gaussian Naive Bayes** (**GNB**) classifiers labeled using different threshold-/**ML**-based labeling strategies against the *Hand-Labeled* and *Hand-Labeled 2019* datasets between July 5th, 2019 and November 8th, 2019.

despite using a different set of features to represent the apps in the training and test datasets. However, unlike the *Drebin* classifier, the **MCC** scores of the **KNN**, **RF**, and **GNB** classifiers seem to be stable across different scan dates, especially on the *Hand-Labeled 2019* dataset. The values

of these scores seem to differ from one classifier to another. On the *Hand-Labeled 2019* dataset, for instance, the average **MCC** scores of the **KNN**, **RF**, and **GNB** classifiers were 0.64, 0.89, and 0.43, respectively. So, it seems that the features used to represent apps in the training dataset and the type of **ML** classifier also impact the performance of a detection method. The results of our experiments show that **Maat**'s **ML**-based labeling strategies that use (selected) naive features, which proved in Section 5.1 to provide the highest **MCC** scores in labeling accuracy, contribute to training different **ML**-based detection methods that are more effective at classifying apps in the *Hand-Labeled* and *Hand-Labeled 2019* datasets as malicious and benign based on vectors of static features extracted from their **APK** archives than detection methods labeled by conventional threshold-based labeling strategies currently adopted within the research community.

## 6 DISCUSSION

In this section, we discuss the insights we gained from our experiments, the limitations of our work, and how we (plan to) address them.

***Optimally Using*** `VirusTotal` ***To Label Apps.*** In Section 1, we mentioned that the main objective of this article is to provide the research community with insights about how to best interpret the raw information given by `VirusTotal` to accurately label Android apps. So, we studied the performance of two types of labeling strategies, viz., threshold-based labeling strategies and a representative of scanner-based labeling strategies that relies on **ML** algorithms trained by our framework **Maat**. During our analysis of `VirusTotal` scan reports, we found that threshold-based labeling strategies that rely on fixed thresholds suffer from two problems. First, they are subjective as their thresholds reflect the subjectivity of defining apps as malicious in terms of the number of scanners deeming it as such. Second, they are susceptible to the dynamicity of `VirusTotal` and its frequent manipulation of scan reports of apps, especially newly developed ones. So, the values of thresholds used by this type of labeling strategies have to be frequently updated to cope with the online platform's dynamicity, rather than being fixed and used for extended periods of time, which can be an infeasible process. We implemented **Maat** to train **ML**-based labeling strategies that are meant to mitigate the aforementioned two problems facing their threshold-based counterparts. While they showed slight sensitivity to `VirusTotal`'s dynamicity (see Section 5.1.2), **Maat**'s **ML**-based labeling strategies—especially those using selected naive features—exhibited steady labeling accuracies over time that mimic that of threshold-based labeling strategies that use the best thresholds at every point in time. Since they are trained at one point in time yet are able to maintain decent labeling accuracies and contribute to more effective **ML**-based detection methods, such **ML**-based labeling strategies have the potential to replace or complement threshold-based labeling strategies.

***Practically Using Maat.*** Given the insights we gained from our experiments, we can recommend using **Maat** to train **ML**-based labeling strategies as follows. We recommend using a dataset of pre-labeled apps and their corresponding `VirusTotal` scan reports in which apps of different classes (e.g., malicious and benign), are **equally distributed**. Having mentioned pre-labeled apps, it is relatively easy to acquire presumably benign apps from platforms, such as *AndroZoo*, in a manner similar to how we acquired apps in the *GPlay* dataset. However, finding pre-labeled, malicious apps is much more difficult, given the difficulty of manually or automatically, yet reliably, labeling apps. Consequently, it is difficult to impose a minimum size of the dataset. We can only recommend gathering as many malicious and benign apps as possible, provided that their ground truth labels are accurate. Given that **Maat**'s labeling strategies are **ML**-based, there are no upper bounds on the size of the dataset. The more important aspect of the dataset, we argue, is that of the age of apps. We recommend using apps that are (a) acquired from **app marketplaces**, and (b) at

least **1 year old**. First, as discussed in Section 4.3, it seems that VirusTotal scanners are oblivious to apps they do not encounter in the wild (e.g., in app marketplaces or in end-user devices). So, self-developed malicious apps are not expected to be labeled as such by an adequate number of scanners, which will provide noisy data points to **Maat** during training. Second, the age of apps is presumed to correlate with the maturity of its VirusTotal scan reports, especially malicious ones. That is, the older the malicious app, the more consensus will be drawn by VirusTotal scanners about it malignancy, which is a process that has been found to take around 1 year after an app has been developed and first seen on VirusTotal [12, 18]. Once the training dataset is gathered, we recommend using **selected naive** features to train **Maat**'s **ML**-based labeling strategies, based on the results of our evaluation in Section 5. Another important aspect of using **Maat** is how often to retrain its **ML**-based labeling strategies. Our experiments show that **Maat**'s **ML**-based labeling strategies continued to accurately label apps based on their VirusTotal scan reports that are 1 year newer than those used to train the labeling strategies. Without showing deterioration in labeling accuracies, we cannot give a solid estimation of the durability of **Maat**'s labeling strategies. However, we can argue that the labeling accuracies of **Maat**'s labeling strategies would start to deteriorate once the content of VirusTotal scan reports and the verdicts of scanners they include significantly differ; for example, if VirusTotal decides to replace the majority of scanners it includes in its scan reports or use inadequate versions of them (e.g., similar to the case of BitDefender). This overhaul of VirusTotal scan reports and the numbers, types, and versions of scanners they include does not regularly occur. Nevertheless, users of **Maat** can regularly test for significant changes in those scan reports using a subset of the training apps themselves. **Maat** users can regularly acquire the latest VirusTotal scan reports of those apps and label them using the trained labeling strategies. If, for a given user of **Maat**, the labeling accuracy significantly drops over a period of time, the user can assume a major change in the scan reports structure that would require **Maat**'s **ML**-based labeling strategies to be retrained.

***The Role of Accurate Labeling in Malware Detection.*** We argued in Section 1 that accurate labeling of apps is fundamental to training and evaluating effective malware detection methods. So, our main hypothesis vis-à-vis labeling is that the more accurately the apps are being labeled, the more effective the malware detection methods that use them as training apps. In Section 5.2, we found that **Maat**'s **ML**-based labeling strategies that use the verdicts of VirusTotal scanners to label apps (i.e., naive features), and threshold-based labeling strategies that use the best current thresholds contribute to training **ML**-based detection methods that are more effective than those labeled using conventional threshold-based strategies that rely on fixed thresholds (e.g., $\texttt{vt} \geq 1$ or $\texttt{vt} \geq 50\%$). Nevertheless, we did not find evidence that accurate labeling of feature vectors is synonymous with better detection accuracy. In some cases, labeling strategies that performed worse at labeling apps based on their VirusTotal scan reports contributed to training **ML**-based detection methods that perform better than other labeling strategies that performed better at labeling apps. We argue that such occasional discrepancies are due to the performance of **ML**-based detection methods being subject to the chosen classifier and the utilized features extracted from the training apps. In general, we found that the more accurate a labeling strategy is at labeling apps based on their VirusTotal scan reports, the more likely it is going to contribute to training more effective **ML**-based detection methods.

**VirusTotal*'s Limitations.*** Despite calls within the research community to replace VirusTotal with a more reliable alternative, the online platform continues to be utilized by researchers to label apps in the datasets they use to evaluate their malware detection methods. These calls for replacement stem from the common knowledge that platforms, such as VirusTotal, suffer from some drawbacks. However, these drawbacks and their impact on the process of labeling

apps were neither thoroughly discussed nor demonstrated. Throughout this article, we discussed some of those limitations and their impacts on identifying the set of correct scanners, estimating the time it takes scan reports to stabilize, and the performance of different labeling strategies. Using the insights we gained from our measurements and experiments, we identified the following limitations of `VirusTotal` that jeopardize its usefulness. Firstly, the platform does not automatically re-scan apps and relies on manually re-scanning apps either via its web-interface or via remote API requests. Secondly, the platform uses scanners or versions of scanners that are not suitable to detect Android malware. Thirdly, for reasons unknown to us, the platform changes the set of scanners it uses to scan the same apps over time, which undermines the sustainability of labeling strategies, such as threshold-based ones. Lastly, the platform does not grant access to the history of scans, effectively preventing researchers from studying the performance of scanners over extended periods of time.

### 6.1 Limitations, Threats to Validity, and Future Work

***Partial Reliance on `VirusTotal`'s Ground Truth.*** To label apps in the *AMD+GPlay* dataset, we relied on the labels generated by Wei et al. to label apps in the *AMD* dataset, which combined filtration of malicious apps using the $\texttt{vt} \geq \texttt{50\%}$ labeling strategy and manual analysis to accurately label apps in the dataset as malicious. We also used the `VirusTotal` scan reports of apps in the *GPlay* dataset, which were downloaded from the well-vetted Google Play store, to deem them as benign according to the criterion *positives==0* between November 2018 and November 8th, 2019. The threat to validity, in this case, is whether those measures we took to ensure the accuracy of the labels in this dataset were not enough. Since we rely on the *AMD+GPlay* dataset to train **Maat**'s **ML**-based labeling strategies, inaccuracies in the labels of those apps threaten to undermine the credibility of our findings. To ensure the credibility of our results, we plan on manually labeling as many Android apps as possible, using them for training **Maat**'s **ML**-based labeling strategies, and comparing the results we achieve with the results in this article. The main obstacle, in this case, is that manually analyzing thousands of apps is a lengthy process.

***Confinement to Used Datasets and Scan Reports.*** The results we recorded and discussed in this article are confined to the datasets that we used and the period within which the `VirusTotal` scan reports were gathered. So, the threat to validity, in this case, is whether the same conclusions we drew can be reached upon using other datasets. For example, would **Maat**'s **ML**-based labeling strategies always manage to mimic the performance of their threshold-based counterparts in terms of accurately labeling apps? In Section 5, in addition to discussing that the structure of **Maat**'s **ML**-based labeling strategies are less susceptible to changes in `VirusTotal`'s scan reports, we attempted to simulate this process examining the performance of **ML**-based and threshold-based labeling strategies over a period of time. However, the experiments were conducted on the same datasets, and the test time period spanned a period of 4 months (i.e., between July 5th, 2019, and November 8th, 2019). As discussed earlier, access to old `VirusTotal` scan reports is only available under commercial licenses, which we consider a limitation of using `VirusTotal` from an academic perspective. Furthermore, beyond November 8th, 2019, we found the academic license we possess, which enabled us to re-scan apps in our datasets and download their up-to-date scan reports, was not granted the re-scan permission anymore. Effectively, comparing the performance of **Maat**'s **ML**-based labeling strategies against threshold-based ones in the future seems to be hindered. To address the first limitation, we plan to use other datasets to run our experiments. This can also be simulated by using random subsets of the *AMD+GPlay* dataset to train **Maat**'s **ML**-based labeling strategies. As for the second limitation, we argue that the adequate method to address this

limitation is to find an alternative to VirusTotal, which includes implementing a platform that mitigates VirusTotal's limitations that we identified in this article.

***Bias of Engineered Features Extracted from* VirusTotal *Reports.*** In Section 5, we found that **Maat**'s **ML**-based labeling strategies trained using naive features extracted from the VirusTotal scan reports of apps in the *AMD+GPlay* dataset are more accurate than their counterparts that are trained using engineered features. After inspecting the importance of individual features in the engineered feature set and their impact on the trained classifiers, we found that some features, such as *positives*, *total*, and *times_submitted*, might introduce bias to the trained classifiers as follows. The *positives* feature, for instance, can effectively segregate malicious and benign apps in the *AMD+GPlay* dataset courtesy of the age of such apps and the maturity of their VirusTotal scan reports, which manifests in those scan reports as higher values of *positives* for malicious apps and lower values for the benign ones. This feature, however, cannot effectively discern the malignancy of newer apps, whose VirusTotal scan reports contains low values of the *positives* attribute as antiviral scanners used by VirusTotal require time to recognize their malignancy. Similarly, the *total* and *times_submitted* attributes do not usually reflect whether an app is malicious or benign. Consequently, engineered features extracted by **Maat** should be designed to (a) accommodate the immaturity of the VirusTotal scan reports of newly developed (malicious) apps, and (b) possibly reflect the malignancy of apps unlike attributes, such as *total*. In future work, we plan to refine the engineered feature set and temporally diversify the training dataset used by **Maat** and study the impact of such adjustments on the structures and labeling accuracies of **ML**-based labeling strategies trained using the refined features.

***Small Size of Test Datasets.*** We base a number of our insights on results achieved against two small datasets, namely, *Hand-Labeled* and *Hand-Labeled 2019*. In particular, these two datasets are used to demonstrate the dynamicity of VirusTotal in Section 4, compare the labeling accuracy of different labeling strategies in Section 5.1, and evaluate the detection capabilities of **ML**-based detection methods labeled using different labeling strategies in Section 5.2. Needless to say, the larger the sizes of these datasets, the more reliable the results based on them will be. In other words, the threat to validity in this case is that our results may not be reliable. The reason behind the relatively small size of the datasets is that we had to manually analyze their apps to ensure accurate ground truth (i.e., malicious and benign), especially since we use those apps to evaluate the accuracy of labeling strategies based on VirusTotal scan reports. Manually labeling thousands of apps is an infeasible process, as discussed in Section 2. To compensate for the size of the datasets, we acquired a random sample of apps from *AndroZoo* that span different ages, sources (i.e., marketplaces), app categories (e.g., games versus utilities), sizes, and so forth. However, aware of the possible limitation, we confine the results of our experiments and measurements to the utilized datasets and aspire to further reinforce them with future work.

***Generalization to other domains, detection methods, and labels.*** In this article, we focused on Android apps, **ML**-based detection methods, and binary labels of malicious and benign. This begs the question of whether we can replicate the same results if we apply evaluate **Maat** against other types of malicious apps (e.g., Windows-based), utilize different features and/or different classifiers, and use more extended labels (e.g., based on malware families or types). The approach that **Maat** adopts, we argue, is domain agnostic and can be applied to other domains, such as Windows-based malware, without any changes to our methods or code base. We only need to acquire datasets of pre-labeled training and test datasets to evaluate **Maat** against, which we plan to do in the future. As for detection methods, there is a plethora of work in this area that spans a multitude of approaches to malware analysis and detection [37], which indeed cannot be comprehensively

covered in this work. So, we plan to conduct more experiments using different types of features (e.g., dynamic features), and/or classifiers to further solidify our findings. Lastly, the problem with using labels based on malware families or types as labels is twofold. First, as we further discuss in Section 7, malware families are subjective labels given by malware analysts to apps and usually differ from one analyst or firm to another. Malware types are equally subjective, especially in cases of malicious apps that exhibit behaviors belonging to two types (e.g., `Worm` and `Ransomware`); some malware analysts might opt to describe the same app as a worm, whereas others prefer to label it as a ransomeware. In either case, we need to find a method to unify those labels prior to training **Maat**. As we discuss in Section 7, there are research efforts that address this problem, which we plan to rely on to extend **Maat**'s capabilities to support more granular labels.

## 7 RELATED WORK

***Studying*** `VirusTotal`. Given the significant role it plays in malware analysis and the detection process, the research community has studied different aspects of `VirusTotal` and its scanners. In [20], Mohaisen et al. inspected the relative performance of `VirusTotal` scanners on a small sample of manually inspected and labeled Windows executables. The authors introduced four criteria, called correctness, completeness, coverage, and consistency, to assess the labeling capabilities of `VirusTotal` scanners and demonstrated the danger of relying on `VirusTotal` scanners that do not meet such criteria. The main objective of this study is, therefore, to shed light on the inconsistencies among `VirusTotal` scanners on a small dataset. In [19], Mohaisen and Alrawi built on their previous study and attempted to assess the detection rate, the correctness of reported labels, and the consistency of detection of `VirusTotal` scanners according to the aforementioned four criteria. They showed that in order to obtain complete and correct (i.e., in comparison to ground truth) labels from `VirusTotal`, one needs to utilize multiple independent scanners instead of hinging on one or a few of them. Similarly, within the domain of Android malware, Hurier et al. studied the scan reports of `VirusTotal` scanners to identify the lack of consistency in labels assigned to the same app by different scanners and proposed metrics to quantitatively describe such inconsistencies [9]. More recently, Peng et al. [23] showed that `VirusTotal` scanners exhibit similar inconsistencies upon deeming **URL** as malicious and benign. The authors also showed that some `VirusTotal` scanners are more correct than others, which requires a strategy to label such **URL**s that does not treat all scanners equally. In Section 4, we discussed how our method **Maat** analyzes `VirusTotal` scan reports to identify correct and stable scanners as part of extracting engineered features from such reports. This process indeed builds on the insights in [9, 19, 20]. In fact, we utilize the *correctness* score introduced in [20] to assess the correctness of `VirusTotal` scanners over time. However, our work is different from the aforementioned work in terms of objectives. While the objective of [9, 19, 20, 23] is to shed light on the dynamicity of `VirusTotal`, our work attempts to take a step further by demonstrating the potential reasons behind `VirusTotal`'s dynamicity and how they impact conventional threshold-based labeling strategies.

The closest work to ours is that by Zhu et al. in [46]. Using the `VirusTotal` scan reports of Windows-based and Android malware gathered daily for a period of almost 2 years, Zhu et al. attempted to answer the following research questions: (a) when can `VirusTotal` labels be considered trustworthy? (b) how are `VirusTotal` scan reports utilized by researchers—according to 115 papers they reviewed—to label apps? and (c) are different `VirusTotal` scanners equally trustworthy? Their findings are similar to ours discussed in Section 4, and Section 5. First, they confirm that `VirusTotal` scanners do flip their verdicts on a regular basis, which undermines the pursuit of a time period within which the `VirusTotal` scan report of any given app is expected to stabilize (see Section 4.2 and Section 4.3). Second, they found that the majority of researchers utilize threshold-based labeling strategies that are heavily impacted by the aforementioned regular flips

of VirusTotal scanners. Third, they found that only a group of VirusTotal scanners provide consistently correct verdicts that reflect the ground truth of apps despite the inconsistencies between the desktop and VirusTotal versions of scanners (see Section 4.2). To the best of our knowledge, the work of Zhu et al. in [46] is the only work that provides the research community with insights similar to the ones we discuss in this article. Their results seem to be obtained independently and concurrently with ours. The work of Zhu et al. is related to ours only in the aspect of studying VirusTotal and its scanners. In addition to studying VirusTotal, its dynamicity, and the impact of this dynamicity on its scanners, our work builds on the insights we gain to devise a framework, **Maat** (see Section 4), that uses VirusTotal scan reports to train **ML**-based labeling strategies that are more resilient to such a dynamicity for longer periods of time.

***Label Unification.*** The lack of universal standards to label and name malicious apps allows different antiviral firms to give different labels to the same malicious app [13, 17]. For example, the same malware can have the labels **Worm**:W32/Downadup.gen!A, Net-**Worm**.Win32.Kido.cp, W32/Conficker.**worm**.gen.a, **Worm**:Win32/Conficker.gen!B, and **Worm**-Win32/Conficker.gen!A, which all share the case insensitive substring *worm* [13]. So, considering the problem to be that of string manipulation, one of the main objectives of academic research has been to devise methods to unify those strings into one that still represents the malware type and family of a malicious app. In [24], Perdisci et al. did not implement a method to unify different antiviral labels; rather, an automated approach, *VAMO*, that assesses methods that clusters similar labels together prior to unifying them. This method can help eliminate noisy labels, which is a prerequisite for the accurate unification of labels. Wang et al. also studied the malware naming discrepancies via analyzing the scan results in [40], and identified two types of such discrepancies, viz., syntactic and semantic. The authors implemented an approach, *Latin*, that considers these two types of discrepancies toward devising a consensus classification of antiviral labels that can be used to look up information about malicious apps in repositories such as *Anubis*. The work in [24] and [40] did not present tools that output unified labels. In [34], Sebastian et al. presented a fully automatic, cross-platform tool, *AVCLASS*, that examines different labels given to the same app by different scanners, and returns the most likely family name that such app should assume. Similarly, Hurier et al. developed a tool, *Euphony*, that mines labels, analyzes the associations between them, and attempts to unify them into common family groups [10]. In this article, we attempt to devise one label for malicious apps based on their VirusTotal scan reports, which include multiple labels given by different antiviral scanners. However, we do not attempt to give a label that indicates the malware family or type of apps. Instead, we attempt to devise labels that indicate whether an app is malicious or benign (i.e., a label that discerns the malignancy of the app).

***Discerning Malignancy.*** A more abstract form of labeling apps is to label them as malicious or benign. Apart from threshold-based labeling strategies, researchers have devised more sophisticated labeling strategies, primarily based on **ML**. In [12], Kantchelian et al. used the VirusTotal scan reports of around 280K binaries to build two **ML**-based techniques to aggregate the results of multiple scanners into a single ground-truth label for every binary. In the first technique, Kantchelian et al. assume that the ground truth of an app (i.e., malicious or benign), is unknown or *hidden*, making the problem of estimating this ground truth that of unsupervised learning. Furthermore, they assumed that the verdicts of more consistent, less erratic scanners are more likely to be correlated with the correct, hidden ground truth than more erratic scanners. Thus, more consistent scanners should have larger weights associated with their verdicts. To estimate those weights and, hence, devise an unsupervised **ML**-based labeling strategy, the authors used an **Expectation Maximization** (**EM**) algorithm based on a Bayesian model to estimate those models. The second technique devised by Kantchelian et al. is a supervised one based on regularized logistic regres-

sion. However, the authors did not describe the nature of the features they use to train such an algorithm. So, we assume that they relied on the verdicts of antiviral scanners in a manner similar to the naive features we extracted from VirusTotal scan reports, as discussed in Section 4. To devise an automated method to label apps based on different verdicts given by antiviral scanners, Sachdeva et al. [26] performed measurements to determine the most correct VirusTotal scanners using scan reports of a total of 5K malicious and benign apps. Using this information, they assign a weight to each scanner that they use to calculate a malignancy score for apps based on their VirusTotal scan reports. Depending on manually defined thresholds, the authors use this score to assign a confidence level of Safe, Suspicious, or Highly Suspicious to test apps. The main difference between our work and [12] and [26] is actionability. That is to say, despite reporting decent labeling accuracies, both efforts did not evaluate whether the newly devised labeling method can help build effective detection methods and left that for future work: "Improved training labels, as obtained by the techniques presented in this paper, should result in an improved malware detector." [12]. In this article, however, we attempted to evaluate the applicability of our method by examining the classification accuracies detection methods can achieve using labels predicted by the most accurate threshold-/**ML**-based labeling strategies.

## 8 CONCLUSION

The infeasibility of manually analyzing and labeling Android apps forces the research community to use VirusTotal scan reports to label those apps using subjective, ad hoc labeling strategies of fixed thresholds. Given its known dynamicity and limitations, VirusTotal needs to be replaced with a more stable platform. However, until this happens, the research community is expected to use VirusTotal. So, the main objective of this article is to provide insights to the research community about how to optimally utilize VirusTotal to scan apps based on their scan reports.

As a step toward standardizing the utilization of VirusTotal scan reports, we developed a method, **Maat**, that automates the process of analyzing VirusTotal scan reports gathered over a period of time to devise **ML**-based labeling strategies that accurately label apps and, in turn, contribute to more effective malware detection. To train such **ML**-based strategies, we analyzed the VirusTotal scan reports of about 53K Android apps gathered between November 2018 and November 8th, 2019 in pursuit of informative features to extract from such reports.

During this process, we addressed issues of interest to the research community (e.g., finding a universal set of correct VirusTotal scanners). In Section 4.2, we showed that there is no universal set of VirusTotal scanners that will always accurately label apps as malicious and benign regardless of the utilized dataset or scan reports. In contrast, we found that the set of correct scanners changes from one dataset to another depending on (a) its composition of malicious and benign apps, and (b) the period during which the apps' scan reports were gathered. We also found that VirusTotal, presumably upon request from the antiviral software companies themselves, replaces adequate versions of some scanners with ones that may not be suitable to detect Android malware, as with the case of BitDefender. The insights we gained from this analysis helped us identify four main limitations of using VirusTotal to label apps, which might aid the research community with developing alternative platforms that mitigate those limitations.

We gained the following insights from **Maat** and its **ML**-based labeling strategies. Firstly, in Section 5.1, we showed that the **ML**-based labeling strategies trained by **Maat** using naive features are more accurate at labeling apps based on their VirusTotal scan reports than their threshold-based counterparts using the currently optimal thresholds of VirusTotal scanners over a period of at least a year. Secondly, we found that **Maat**'s **ML**-based strategies contributed to training **ML**-based detection methods, such as the *Drebin* classifier [3], that are more effective at detecting out-of-sample Android malware. In summary, until VirusTotal's limitations are either addressed

or new platforms are developed, we believe that **Maat** trains labeling strategies that provide a viable alternative to unsustainable threshold-based strategies.

## REFERENCES


[1] 2019. K-9 Mail—Advanced Email for Android. Retrieved on June 2019 from http://tiny.cc/1fbb7y.

[2] 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 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR'16)*. IEEE, 468–471.

[3] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. DREBIN: Effective and explainable detection of android malware in your pocket. In *NDSS*.

[4] Saba Arshad, Munam Ali Shah, Abid Khan, and Mansoor Ahmed. 2016. Android malware detection and protection: A survey. *International Journal of Advanced Computer Science and Applications* 7 (2016), 463–475.

[5] Davide Chicco and Giuseppe Jurman. 2020. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics* 21, 1 (2020), 6.

[6] Ken Dunham, Shane Hartman, Manu Quintans, Jose Andre Morales, and Tim Strazzere. 2014. *Android Malware and Analysis*. Auerbach Publications.

[7] Roberto Jordaney Johannes Kinder Feargus Pendlebury, Fabio Pierazzi and Lorenzo Cavallaro. 2019. TESSERACT: Eliminating experimental bias in malware classification across space and time. In *28th USENIX Security Symposium*. USENIX Association, Santa Clara, CA.

[8] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Ainuddin Wahid Abdul Wahab. 2015. A review on feature selection in mobile malware detection. *Digital Investigation* 13 (2015), 22–37.

[9] Médéric Hurier, Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. On the lack of consensus in anti-virus decisions: Metrics and insights on building ground truths of android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 142–162.

[10] Médéric Hurier, Guillermo Suarez-Tangil, Santanu Kumar Dash, Tegawendé F. Bissyandé, Yves Le Traon, Jacques Klein, and Lorenzo Cavallaro. 2017. Euphony: Harmonious unification of cacophonous anti-virus vendor labels for android malware. In *Proceedings of the 14th International Conference on Mining Software Repositories*. IEEE Press, 425–435.

[11] AV-Test: The Independent IT-Security Institute. 2019. The best antivirus software for Android. Retrieved on June 2019 from http://tiny.cc/1k66az.

[12] Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Brad Miller, Vaishaal Shankar, Rekha Bachwani, Anthony D. Joseph, and J. Doug Tygar. 2015. Better malware ground truth: Techniques for weighting anti-virus vendor labels. In *Proceedings of the 8th ACM Workshop on Artificial Intelligence and Security*. ACM, 45–56.

[13] Tom Kelchner. 2010. The (in) consistent naming of malcode. *Computer Fraud & Security* 2010, 2 (2010), 5–7.

[14] Li Li, Daoyuan Li, Tegawendé F. Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro. 2017. Understanding android app piggybacking: A systematic study of malicious code grafting. *IEEE Transactions on Information Forensics and Security* 12, 6 (2017), 1269–1284.

[15] Symphony Luo and Peter Yan. 2014. Fake Apps: Feigning Legitimacy. Retrieved on April 2019 from https://goo.gl/hzZBiH.

[16] PC Magazine. 2008. BitDefender Free Edition. Retrieved on June 2019 from http://tiny.cc/8vx9az.

[17] Federico Maggi, Andrea Bellini, Guido Salvaneschi, and Stefano Zanero. 2011. Finding non-trivial malware naming inconsistencies. In *International Conference on Information Systems Security*. Springer, 144–159.

[18] Brad Miller, Alex Kantchelian, Michael Carl Tschantz, Sadia Afroz, Rekha Bachwani, Riyaz Faizullabhoy, Ling Huang, Vaishaal Shankar, Tony Wu, George Yiu, et al. 2016. Reviewer integration and performance measurement for malware detection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 122–141.

[19] Aziz Mohaisen and Omar Alrawi. 2014. AV-meter: An evaluation of antivirus scans and labels. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 112–131.

[20] Aziz Mohaisen, Omar Alrawi, Matt Larson, and Danny McPherson. 2013. Towards a methodical evaluation of antivirus scans and labels. In *International Workshop on Information Security Applications*. Springer, 231–241.

[21] Jon Oberheide and Charlie Miller. 2012. Dissecting the android bouncer. *SummerCon2012, New York*.

[22] Feargus Pendlebury, Fabio Pierazzi, Roberto Jordaney, Johannes Kinder, and Lorenzo Cavallaro. 2018. Enabling fair ML evaluations for security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2264–2266.

[23] Peng Peng, Limin Yang, Linhai Song, and Gang Wang. 2019. Opening the blackbox of virustotal: Analyzing online phishing scan engines. In *Proceedings of the Internet Measurement Conference*. ACM, 478–485.

[24] Roberto Perdisci et al. 2012. VAMO: Towards a fully automated malware clustering validity analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*. ACM, 329–338.
[25] Neil J. Rubenking. 2015. False Positives Sink Antivirus Ratings. Retrieved on June 2019 from http://tiny.cc/eyh2uz.
[26] Shefali Sachdeva, Romuald Jolivot, and Worawat Choensawat. 2018. Android malware classification based on mobile security framework. *IAENG International Journal of Computer Science* 45, 4 (2018).
[27] Aleieldin Salem and Alexander Pretschner. 2018. Poking the bear: Lessons learned from probing three android malware datasets. In *Proceedings of the 1st International Workshop on Advances in Mobile App Analysis*. ACM, 19–24.
[28] Hillary Sanders and Joshua Saxe. 2017. Garbage in, garbage out: How purportedly great ML models can be screwed up by bad data. Technical Report.
[29] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, and Pablo Garcia Bringas. 2012. On the automatic categorisation of android applications. In *Proceedings of the 2012 IEEE Consumer Communications and Networking Conference (CCNC'12)*. IEEE, 149–153.
[30] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. 2013. Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions*. Springer, 289–298.
[31] Ryo Sato, Daiki Chiba, and Shigeki Goto. 2013. Detecting android malware by analyzing manifest files. *Proceedings of the Asia-Pacific Advanced Network* 36 (2013), 23–31.
[32] scikit learn. 2019. Feature Selection. Retrieved on October 2019 from http://tiny.cc/2d997y.
[33] scikit learn. 2019. GridSearchCV. Retrieved on October 2019 from http://tiny.cc/mquy9y.
[34] Marcos Sebastián, Richard Rivera, Platon Kotzias, and Juan Caballero. 2016. AVclass: A tool for massive malware labeling. In *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 230–253.
[35] Tim Stazzere. 2016. Detecting pirated and malicious android apps with apkid. Retrieved from https://goo.gl/Na2ZAX.
[36] Guillermo Suarez-Tangil, Santanu Kumar Dash, Mansour Ahmadi, Johannes Kinder, Giorgio Giacinto, and Lorenzo Cavallaro. 2017. DroidSieve: Fast and accurate classification of obfuscated android malware. In *Proceedings of the 7th ACM Conference on Data and Application Security and Privacy*. ACM, 309–320.
[37] Kimberly Tam, Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Lorenzo Cavallaro. 2017. The evolution of android malware and android analysis techniques. *ACM Computing Surveys (CSUR)* 49, 4 (2017), 76.
[38] VirusTotal. 2019. VirusTotal. Retrieved on September 2019 from http://tiny.cc/xjbb7y.
[39] Haoyu Wang, Zhe Liu, Jingyue Liang, Narseo Vallina-Rodriguez, Yao Guo, Li Li, Juan Tapiador, Jingcun Cao, and Guoai Xu. 2018. Beyond Google Play: A large-scale comparative study of Chinese android app markets. In *Proceedings of the Internet Measurement Conference 2018*. ACM, 293–307.
[40] Ting Wang, Shicong Meng, Wei Gao, and Xin Hu. 2014. Rebuilding the tower of babel: Towards cross-system malware information sharing. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*. ACM, 1239–1248.
[41] Fengguo Wei, Yuping Li, Sankardas Roy, Xinming Ou, and Wu Zhou. 2017. Deep ground truth analysis of current android malware. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 252–276.
[42] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *Proceedings of the 2012 7th Asia Joint Conference on Information Security (Asia JCIS'12)*. IEEE, 62–69.
[43] 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*. ACM, 288–302.
[44] Wu Zhou, Yajin Zhou, Michael Grace, Xuxian Jiang, and Shihong Zou. 2013. Fast, scalable detection of piggybacked mobile applications. In *Proceedings of the 3rd ACM Conference on Data and Application Security and Privacy*. ACM, 185–196.
[45] Yajin Zhou and Xuxian Jiang. 2012. Dissecting android malware: Characterization and evolution. In *2012 IEEE Symposium on Security and Privacy (SP'12)*. IEEE, 95–109.
[46] Shuofei Zhu, Jianjun Shi, Limin Yang, Boqin Qin, Ziyi Zhang, Linhai Song, and Gang Wang. 2020. Measuring and modeling the label dynamics of online anti-malware engines. In *29th USENIX Security Symposium (USENIX Security 20)*.

Received July 2020; revised April 2021; accepted May 2021