Hindawi
Security and Communication Networks
Volume 2018, Article ID 5749481, 11 pages
https://doi.org/10.1155/2018/5749481

Research Article

Android Malware Characterization Using Metadata and Machine Learning Techniques

Ignacio Martín,[1] **José Alberto Hernández**,[1] **Alfonso Muñoz**,[2] **and Antonio Guzmán**[2]

[1] *Universidad Carlos III de Madrid, Spain*
[2] *Telefónica Digital Identity & Privacy, Spain*

Correspondence should be addressed to Ignacio Martín; ignmarti@it.uc3m.es

Received 13 February 2018; Revised 4 June 2018; Accepted 19 June 2018; Published 8 July 2018

Academic Editor: Sherali Zeadally

Android malware has emerged as a consequence of the increasing popularity of smartphones and tablets. While most previous work focuses on inherent characteristics of Android apps to detect malware, this study analyses indirect features and metadata to identify patterns in malware applications. Our experiments show the following: (1) the permissions used by an application offer only moderate performance results; (2) other features publicly available at Android markets are more relevant in detecting malware, such as the application developer and certificate issuer; and (3) compact and efficient classifiers can be constructed for the early detection of malware applications prior to code inspection or sandboxing.

1. Introduction and Motivation

The mobile market industry has explosively grown in the last decade. According to the latest estimates, the number of smartphone users has reached 2 billion at the beginning of 2016 and is expected to grow up to more than 2.50 billion in 2018 (see https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide, last access June 2018).

Android has positioned itself as the leading operating system in the smartphone industry, accounting for more than 81% of devices by the end of 2016 (see http://www.idc.com/prodserv/smartphone-os-market-share.jsp, last access June 2018). Indeed, one key for its success is that the Android platform is open to any developer, individual, or enterprise that is able to easily design new applications and services and upload them to any of the Android markets available: Google Play Store, Amazon Appstore, Samsung Galaxy Apps, etc. At the time of writing, it is estimated that nearly 2.7 million applications are uploaded at Google Play, while new applications are uploaded at a pace of more than 60 thousand per month (see http://www.appbrain.com/stats/number-of-android-apps, last access June 2018).

Unfortunately, the popularity of Android and the facilities it provides to develop and upload applications have side effects. Furthermore, the variety of Android markets favors the existence of rogue markets where applications follow not-so-stringent reviews and have propitiated even more the development of a large malware ecosystem. In this light, Android has become one of the most valuable targets for malware developers. An extensive taxonomy of Android malware applications, where up to 49 malware families have been identified, can be found in [1]. In general, there has been a great effort in the development of software security tools capable of dealing with the continuously growing malware ecosystem and rogue applications; despite that most of these efforts have been focused on code-based analysis.

However, a good deal of information is already available as metadata at Google Play and can be used to identify patterns not yet pointed out in previous work, as far as we are concerned. Application information like the name of its developer, its category, the number of downloads, and the number of votes received have not been studied in the past to identify malware patterns. Such *metadata* provides a good ground for static malware detection which does not require behaviour analysis and provides a fast first-stage notion on whether an application *"behaves suspiciously"* (shows malware patterns) or not.

To this end, this work focuses on the analysis of such indirect features and their ability to unveil malware. We analyse metadata to find a subset of features which have proven predictive power and use them to develop and test different machine learning (ML) models. Specifically, the main contributions of this work are

(i) analysing and assessing Android metadata and permissions as effective malware predictors,

(ii) proposing a machine learning malware detection model that relies on metadata information publicly available at Google Play,

(iii) evaluating such model and assessing its potential as a first-stage malware filter to detect Android malware.

2. Related Work

The ability to early detect malicious Android applications is vital to enhance user security, since Android apps can be tagged, reported, and removed from the market and their signatures can be blacklisted. This can be seen as a classification problem and, therefore, many authors have attempted to use machine learning over diverse Android-application-based feature sets.

In fact, a survey on machine learning techniques applied to malware detection may be found in [16]. For instance, the authors in [2] gather features from application code and manifest (permissions, API calls, etc.) and use Support Vector Machines (SVMs) to identify different types of malware families. The authors in [3] analyse Bayesian-base machine learning techniques for Android malware detection. In [4], the authors use permissions and control flow graphs along with Support Vector Machines (SVMs) to differentiate malware from good applications ("goodware" in what follows). The authors in [5] use API calls and permissions as features to train SVMs and Decision Trees (DTs). Androdialysis [6] explores the intents of each application as features for the classification task. Yerima et al. [7] try different algorithms over API calls and command sets and show promising results for ensemble methods, such as Random Forests (RFs).

In general, Android permissions have been extensively studied under the assumption that these are critical in identifying malware; see [8, 17–19]. Actually, in [8] the authors discover that malware applications use less permissions than goodware ones.

The authors in [9] attempt malware detection by inspecting other application run-time parameters, such as CPU usage, network transmission, and process and memory information. Mas'ud et al. [20] also include Android system calls in the detection strategy. Furthermore Elish et al. [10] propose a single-feature classification system based on user behaviour profiling. DroidChain authors [11] propose a novel model which analyses static and dynamic features of applications assuming different malware models. Recently, VirusTotal has released *Droidy* [21], a sandbox system capable of extracting information regarding malware samples such as Network and SMS activities, Java reflection calls, and filesystem interaction.

In a different approach, the authors of [15] design a differential-intersection analysis technique to identify repackaged versions of popular applications, which is a common way to disguise malicious applications.

Concerning malware detection systems, there exist two main trends: (1) online services which aim to procure efficient and lightweight solutions to cope with malware detection from the mobile device and (2) offline services to engage in fast analysis of large amounts of applications in order to mark potentially harmful code, either for removal or extended inspection. In this light, several authors have explored both trends, obtaining results such as the systems exposed in [2, 12, 22] which provide online solutions to inform or warn the user on the device or more general, hardware-dependent systems such as [13, 14] which are scalable systems capable of dealing with huge amounts of applications at once, enabling fast and cheap detection mechanisms for entities like application markets to improve the quality of their apps. The authors of [23] extensively survey the types and works regarding malware detection system.

In addition, obtaining as much information as possible on threats and other undesired applications is really necessary, and various authors propose methodologies and systems to collect diverse and huge amounts of data. For example, Burguera et al. [24] propose a framework for collecting application trace and identify uncommon behaviours of common applications. Moreover, the authors of [25, 26] propose a system to gather signatures and malware information automatically.

Finally, Table 1 summarises strong and weak points of different works in the literature together with their reported performance. Although many approaches obtain very high accuracy rates, they mainly require the apk file and code inspection to perform their analysis. Oppositely, our approach based on metadata focuses on a novel feature set consisting in publicly available metadata, allowing a simpler approach to malware detection. Indeed, that feature set has not been used before; only the authors in [27] partially addressed metadata by performing sentiment analysis over users' comments in Android applications.

The remainder of this work is organized as follows: Section 3 describes the dataset under study, including number of applications and types of features analysed. Section 4 explains the methodology, whereas Section 5 reports the experiments and results obtained. Finally, Section 6 concludes this work with a summary of the findings.

3. Dataset Description and Preprocessing

Table 2 provides a summary of the dataset used in this article. The dataset comprises around 118 thousand Android applications collected from Google Play Store during year 2015. This dataset has been obtained using the Tacyt cyber-intelligence tool developed internally at Eleven Paths (Telefónica Group; see Acknowledgments for further details). For each application, we have extracted not only intrinsic features of the Application PacKage (apk) file, e.g., size in bytes or list of permissions used, but also other metadata available at Google Play, including that related to the application developer or

Table 1: Detailed summary of previous malware detection systems in Android.

Title	Target	Pros	Cons	Dataset	Performance
Drebin [2]	On-device malware detection using code and manifest permissions	On-device; Explainability;	Obfuscation; Accuracy for imbalanced	123, 453	*Accuracy*: 0.94
Yerima et al. [3]	Bayesian analysis over permissions and other code features for zero-day analysis	Simple model; High accuracy	Small sample set; Code inspection	1, 000	*Accuracy*: 0.93
Sahs et al. [4]	Use of SVM over permissions and control flow graphs for malware detection	Anomaly detection- based approach	Unbalanced results; Offline system	-	*F1-score*: 0.25
Peiravian et al. [5]	Tree and SVM Classifiers over permissions and API Calls	High accuracy; robust modelling	Code analysis; Unknown processing times; API calls can be benign	3, 000	*F1-score*: 0.95
AndroDialysis [6]	Analysis of application intents and permissions as features for machine learning	High accuracy; On-device	Code analysis; Unbalanced dataset	7, 406	*Accuracy*: 0.955
Yerima et al. [7]	Ensemble learning over app code and API calls	High accuracy; Feature Selection not needed	Code analysis; Large-feature model	6, 863	*Accuracy*: 0.97
PUMA [8]	Permission Analysis	Simple approach; Robust modelling	Tiny dataset; Narrow feature set	239	*AUC*: 0.92
Ham et al. [9]	Classifier using app's runtime parameters	predictive features; High F1-score	Code analysis; Unknown processing times	14, 794	*F1-score*: 0.99
Karim et al. [10]	User profile single feature classification	Compact model; low False Positive rate;	Complex feature extraction; User profiles can be fooled	4, 117	False Positive Rate: 2.1%
DroidChain [11]	Behaviour chain detection of malware	Zero-day support; Behaviour detection	Code analysis; Restricted to 4 malware models	1, 260	*F1-score*: 0.8
Andromaly [12]	Application monitoring and anomaly detection	distributed service; On-device	Monitoring impact on device; Lack of Ground truth; May require device rooting	20	*AUC*: 0.8-0.99
RiskRanker [13]	Zero-day malware detection through automated code analysis	Large Dataset; Multi-stage; Risk assessment	Code analysis; Limited support of behaviours; Signature dependent	118, 318	*Number of Detections*: 718
DroidAPIMiner [14]	KNN classifiers over bytecode semantic information	Fast solution; Family-aware system	Code analysis; API calls can be benign; Static Whitelisting	20,000	*Accuracy*: 0.99
Massvet [15]	Application code differential analysis to spot clones by comparing with market	Very Large dataset; Scalable; Pairwise app comparison	Vulnerable to code changes; Code analysis	1, 2M	*Number of Detections*: 127, 429

Table 2: Dataset overview. Date of collection: 2015.

Dataset element	Dataset Figures
Number of Applications	118, 846
Applications flagged as malware	69, 918
Number of Single-detection applications	34, 025
Number of Developers	53, 780
Number of Certificate Issuers	44, 244
Number of different permissions	21, 541
Number of intrinsic features	15
Number of social-related features	7
Number of reputation features	2

the number of votes or average star rating (see Table 3 for an overview of the metadata extracted).

Next section overviews the features derived from such data; some of them will be extremely powerful in identifying potential malware.

3.1. Intrinsic Application Features. These relate to concise application information, including its size (bytes), application category, and number of images and files used by the application. This group comprises 14 features.

Other intrinsic features considered in the analysis include the permissions used by each apk. There are over 21K different permissions used by the applications in our dataset; most popular ones are

(i) android.permission.internet (found in 96.07% of apps),

(ii) android.permission.access_network_state (91.15%),

(iii) android.permission.read_external_storage (54.5%),

(iv) android.permission.write_external_storage (54.12%),

(v) android.permission.read_phone_state (39.81%).

Many permissions appear only once in the dataset as they are often self-defined permissions. Thus, the binarized permission features comprise a very-sparse high-dimensional matrix. In these cases, feature hashing [28] is an effective strategy for dimensionality reduction; it works by grouping applications according to some hash functions. We will leverage the hashing trick in the paper to reduce the number of intrinsic application features as compared to using permissions in their raw form.

3.2. Social-Related Features. These are 7 features and involve feedback collected from users in the market. As Google Play is strongly connected with the social network Google+, features like total number of votes or average rating are provided. For each possible ratings (1, 2, 3, 4, and 5 stars) we acquire the number of votes given. Then, it is possible to easily compute the mean average of any application in the market as well as the total number of votes for that application.

3.3. Entity-Related Features: Developers and Certificate Issuers. Android markets often provide information about the application developers (name, email address, website, etc.) and the certificate information of the application signature (expedition or expiration dates, issuer or subject names, etc.).

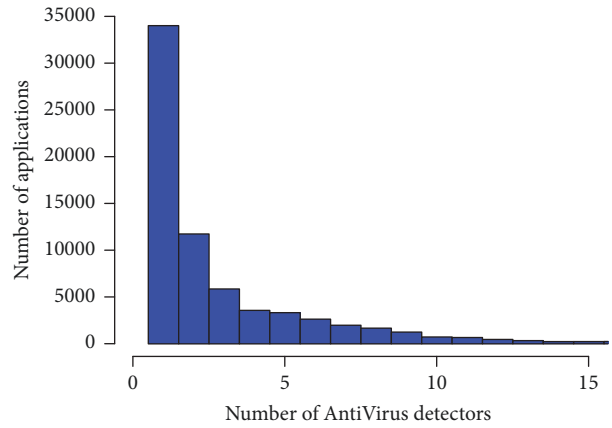


Figure 1: Histogram of AV detectors per malware application.

In our dataset, there are around 53K different developer names and 44K certificate issuer names. The reader must note that Google Play allows self-signed applications, i.e., applications where the issuer is the same as the developer. As a result, in many cases, the issuer of a certificate and the developer may be the same entity. However, their reputations may change, as many issuers may not only sign their own applications and not all developers self-sign their applications (and even if they do, they use different accounts).

Following [29], we have created two new features called developerRep and issuerRep which account for the percentage of applications that each developer and certificate issuer have tagged as malware. These metrics are computed during the training phase of the ML algorithm with information available from the training data; in other words, the test set is never used in the computation of this metrics.

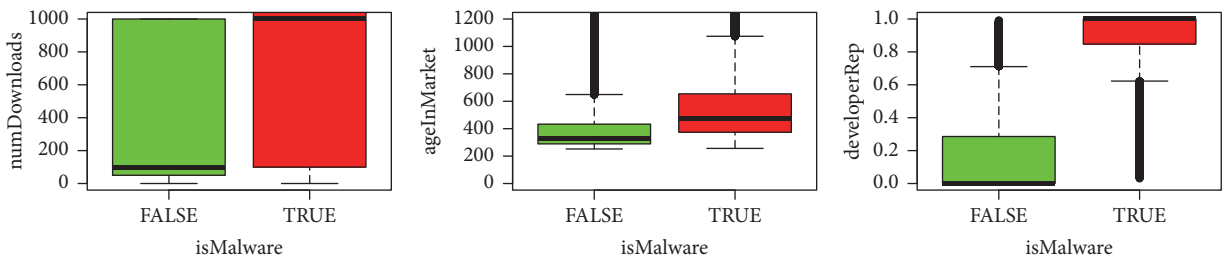
3.4. Malware Detection Attributes. Once downloaded, all applications have been inspected for malware using the VirusTotal web service (free online virus, malware, and URL scanner, available at http://www.virustotal.com/, last access June 2018). VirusTotal checks each application against a large number of malware engines, producing a binary result (malware/goodware) per engine (McAfee, AVG, VIPRE, TrendMicro, etc.). In our dataset, around 69K applications have been declared as malware by at least one of these engines.

Concerning the number of detectors per malware application, a Zipf-like behaviour is observed; i.e., most malware applications are only detected by a single antivirus (AV) engine, while a few number of malware applications are detected by many AV engines. In particular, 25% of the malware applications are detected by 1 AV engines or less (1st Quartile), 50% are detected by 2 AV engines or less (median), and 75% malware applications are detected by 4 AV engines or less (3rd Quartile). We shall use the label "isMalware" (TRUE/FALSE) to denote whether an application is tagged as malware or not.

Figure 1 shows a histogram of the frequency of each application detection count. The Zipf-like behaviour is clear in the picture, as most applications are only detected by

Table 3: Summary of features, description, and values of the metadata collected from Google Play.

	Name	Description	Value
	Intrinsic features		
1	size	Application size in bytes	Numeric
2	categoryName	Assigned Google Play Category	Categoric
3	ageInMarket	Number of days the app has been on Google Play	Numeric
4	lastSignatureUpdate	Number of days from last app signature update	Numeric
5	timeFromCreation	Number of days since the application was developed	Numeric
6	lastUpdate	Number of days since the application was last updated	Numeric
7	certVal	Number of days from which application is valid	Numeric
8	oldestDateFile	Number of days from the creation of the oldest file in the application	Numeric
9	numPerm	Total number of permissions required by the application	Numeric
10	numFiles	Total number of files the application contains	Numeric
11	numImages	Total number of images the application contains	Numeric
12	numDownloads	Total number of times the application has been uploaded	Numeric
13	versionCode	Google Play reported version of the application	Numeric
14	f+number features	Each of the different Feature hashes	Numeric
	Social-related features		
15	totalVotes	Total number of rating votes given to the application	Numeric
16	OneStarRatingCont	Number of one-star votes received	Numeric
17	twoStarRatingCont	Number of two-star votes received	Numeric
18	threeStarRatingCont	Number of three-star votes received	Numeric
19	fourStarRatingCont	Number of four-star votes received	Numeric
20	fiveStarRatingCont	Number of five-star votes received	Numeric
21	meanStar	weighted average rating of the application	Numeric
	Entity-related features		
22	developerRep	Developer reputation metric	Numeric
23	issuerRep	Issuer reputation metric	Numeric
	Label		
L	isMalware	True if flagged by one or more AV engines	Boolean



(a) Number of downloads (b) Number of days in Google Play (c) Associated developer reputation

Figure 2: Goodware/malware boxplot comparison for three features: number of downloads, number of days since the application was uploaded, and developer reputation.

a single engine (34, 025 applications), while the average detection count is 3. Furthermore, there is one application detected as malware by as many as 53 AV detectors.

Due to this disparity and disagreement among AVs, we will consider the aforementioned quantiles (1-AV, 2-AV, and 4-AV detection) as different thresholds to establish ground truth rules within the detection scheme.

In summary, Table 3 shows a comprehensive description of all features in the dataset, their description, and the type of variable they are.

4. Methodology and Data Analysis

4.1. Initial Approach. Feature selection is key to reduce complexity and improve performance. We expect some features to have more predictive power than others, as noted in Figure 2. In this figure, three boxplots for malware/goodware classes are shown for three sample features: the number of times the application has been downloaded from the market (Figure 2(a)), the time the application has been in Google Play (Figure 2(b)), and the developer reputation (Figure 2(c)).

As observed, the number of downloads is not a very useful feature, since both goodware and malware show similar 25-percentile (around 10) as well as 75-percentile (48) values. Concerning the number of days in Google Play (centre), the 25-, 50-, and 75-quantile measures of malware differ from goodware, showing some predictive power. Finally, developers reputation (Figure 2(c)) clearly reveals that malware developers tend to develop more malware while goodware developers create almost no malware.

4.2. Classification Models and Performance Evaluation. In a binary classification problem, we are often given a training set with labeled data $\{X_i, y_i\}_{i=1}^{N_{tr}}$, where $y_i \in \{0, 1\}$ and X_i is a vector containing the values of P predictors or features, namely, $X_i = (x_{i1}, \ldots, x_{iP})$. In our case, the labels y refer to the categoric variable "*isMalware*", whereas the predictors X_i comprise 512 feature hashes of permissions, 15 intrinsic features, 7 social-related features, and both Issuer and Developer reputations.

Machine learning algorithms are in charge of constructing a function $g(X)$ from the training set that separates the two classes with minimum error. In our experiments, we have used *logistic regression (LR)*, *Support Vector Machines (SVMs)*, and *Random Forests (RF)* as three well-known supervised ML algorithms.

Once a model is obtained, the following stage comprises testing its ability to predict the result of unobserved data samples, i.e., evaluating the model's generalization capabilities. Tenfold cross-validation has been used to evaluate test error, measured using well-known metrics: *Receiver Operating Characteristic (ROC) curves* and the *Area Under ROC Curve (AUC-ROC)*, *Precision*, *Recall*, and *F1-score*.

Regarding model's intrinsic *hyperparameters*, tenfold cross-validation over the training sets has been used. In other words, at each iteration, the training data is divided again in 10-fold used to find optimal hyperparameter tuning using the well-known *Grid Search* strategy.

4.2.1. Validation and Significance. Tenfold cross-validation consists in splitting the entire dataset in 10 chunks of equal size and perform 10 iterations over them, selecting at each turn a different chunk to be the testing set and the reminding ones to be the training. Using this method, one can perform hyperparameter tuning, but also provide results with statistical significance (i.e., robust results which do not depend on the selection of training/test instances).

4.3. Feature Selection. Some features are critical in the discrimination of good/malware while others are not, either due to correlation or small predictive power. For selecting the most relevant features, we have used the following methods.

4.3.1. Pearson's Chi Squared Test. Statistical test used to determine whether any difference among variables occurs by chance or there is indeed a statistical relation.

4.3.2. Entropy-Based Methods. In information theory, entropy measures the amount of unknown information a certain source provides. The following measurements are considered:

(i) *Information Gain (IG)* or mutual information between a feature X_i and the outcome y.

(ii) *Gain Ratio (GR)* is the result of dividing the information gain by the intrinsic information of the feature, aiming at reducing bias towards features with high information gain value on its own rather than a good relationship with the output variable y.

4.3.3. Random Forest Importance. Random forest importance refers to the contribution of each node in the algorithm. In particular, we consider the Mean Decrease in Node Impurity, which measures how unequal the nodes in each tree of the forest are.

For further reference of machine learning and statistical methods for data analysis, the reader is referred to [30].

5. Experiments and Results

In the experiments, we have used the well-known R open-source statistical software, along with a number of libraries for machine learning and feature selection (MASS, randomForest, kernlab, glmnet, mlr, and caret). From the original dataset, we have built nine different subsets of 50K apps with different compositions. Concisely, for each subset we vary either the amount of malware it contains (2%, 25%, or 50% of malware over the total) or the threshold used for considering an application as malware (1-AV, 2-AV, and 4-AV detection). As an example, we shall refer to the (1-AV, 25%) malware dataset as a dataset that contains 25% malware and 75% goodware applications where malware is randomly selected among all applications whereby at least 1-AV detector was fired.

There is an exception: the (4-AV, 50%) dataset. This dataset contains 36K samples as a result of the lack of malware applications detected as such by more than 4 AV engines.

5.1. Predictive Power of Permissions. As noted in the introduction, several researchers have studied the permissions used by an application and their ability to detect malware. For instance, the authors in [31] achieve F1-score values in the range of 0.6 to 0.8.

In order to evaluate the effects that feature hashing has on permissions, we try different hashing spaces (32, 64, 128, 256, 512, 1024, and 2048 hashes) to evaluate the feature amount-performance trade-off. To measure performance, we run 10-fold cross-validation for threshold tuning in a logistic regression algorithm and compute different AUC (Area Under the Curve) measurements for each of the hashing spaces.

In our case, Figure 3 shows the ROC curve and AUC-ROC values using logistic regression with different number of hashes for the (4-AV, 50%) dataset. As observed, the more hash functions used, the higher AUC achieved in the range of 0.7 for 256 hashes and above, in line with [31]. In conclusion, the permissions set alone offer a moderate approach to detect Android malware.

Hence, we choose 512 hashes as a good trade-off between model accuracy and the number of features introduced, as

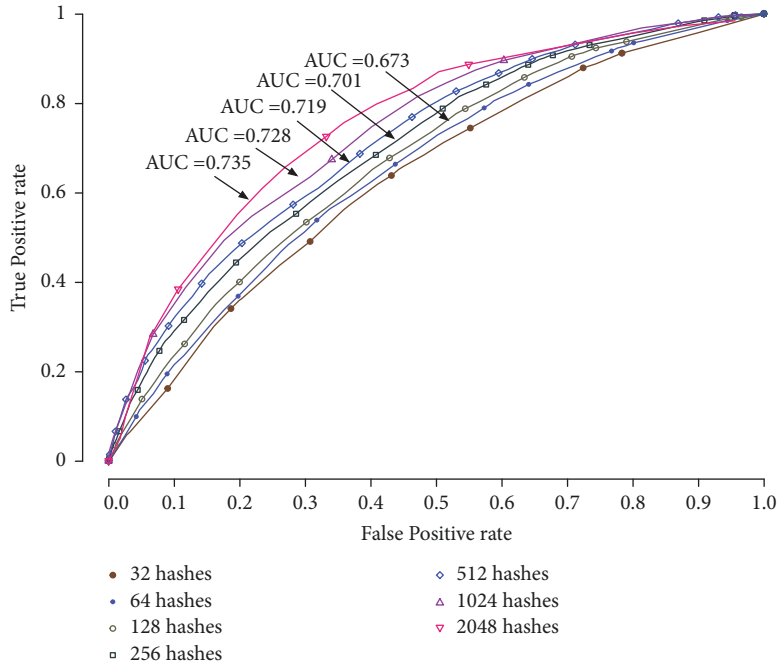


Figure 3: ROC curve for malware detection using feature hashing on permissions only.

more features improve performance at the cost of considerably larger complexity. In this case, average F1-score is 0.675, whereas the area under the Precision-Recall (PR) curve is 0.685 for logistic regression. Regarding other algorithms, 512 feature hashes on their own achieve F1-score values of 0.659 for SVMs and 0.653 for RFs.

In the next sections we study the remaining 26 metadata features (i.e., intrinsic, social, and entity-related) along with 512 feature hashes and apply feature selection techniques to identify the most relevant ones.

5.2. Feature Selection. Beginning at 535 features in the dataset, variable selection is performed to reduce model complexity. Generally, larger predictor collections do not necessarily imply better performance but larger complexity. In fact, the more predictors considered, the easier to bump into the well-known "Curse of dimensionality", which occurs when there is a large proportion of predictors with respect to data, penalizing global performance.

In the first experiment, Figure 4(a), we have used the four feature selection methods described in Section 4 to evaluate the importance of each feature in the dataset. The results show such features sorted by each selection index and normalized with respect to the largest (names of features are self-explanatory). This experiment was conducted using the (4-AV, 50%) dataset.

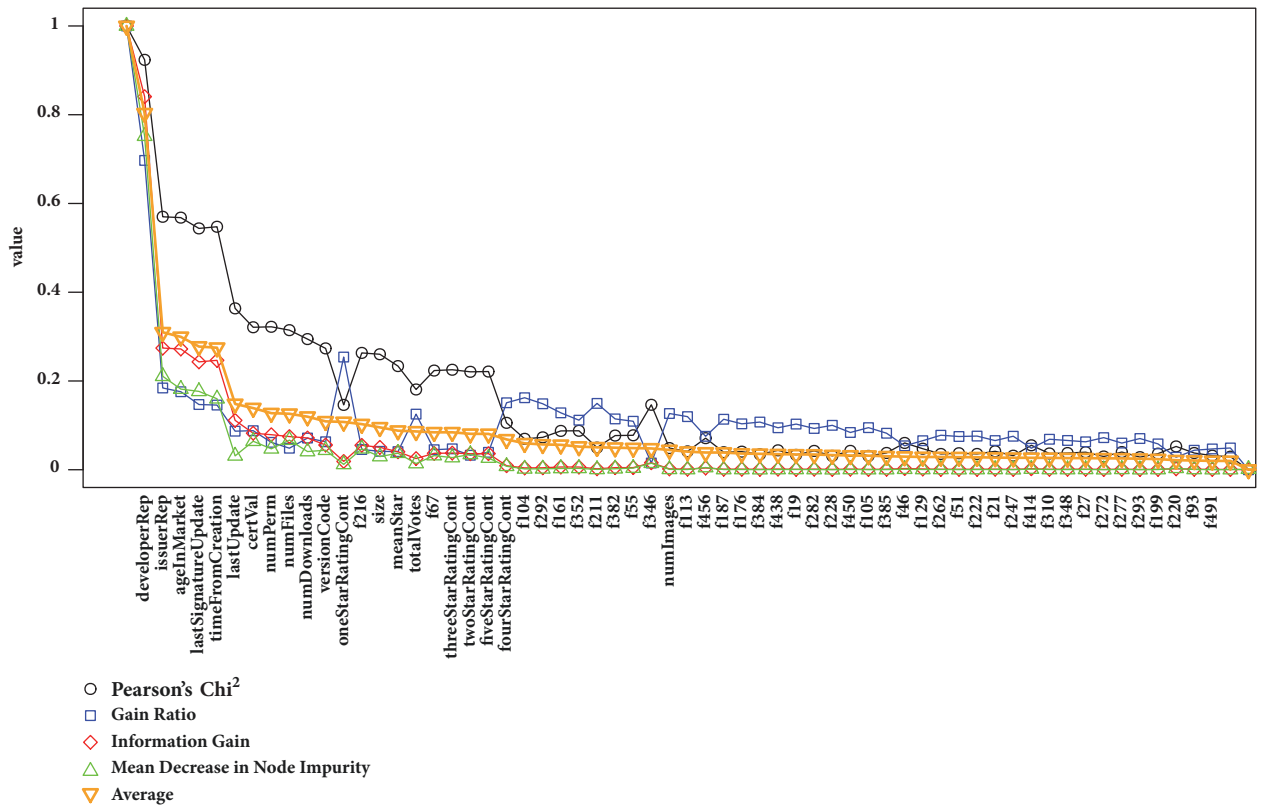
As shown in Figure 4(a), the top-7 most relevant features in the dataset are, in order of importance, developerRep, issuerRep, ageInMarket, lastSignatureUpdate, timeForCreation, lastUpdate, and certVal (see Table 3 for a description of them). In contrast, the feature hashes on the permissions are not relevant when compared with the others.

In order to establish the number of valid features for modeling, Figure 4(b) shows the tenfold cross-validated F1-score versus the number of predictors involved for each algorithm (RF, LR, and SVM), where new predictors are added at each iteration in decreasing order of relevance. There, Random Forest provides the highest F1-score (around 0.89), while LR and SVM reach around 0.86 and 0.87, respectively. Moreover, the figure shows that highest performance on any algorithm may be achieved with only the top-15 features.

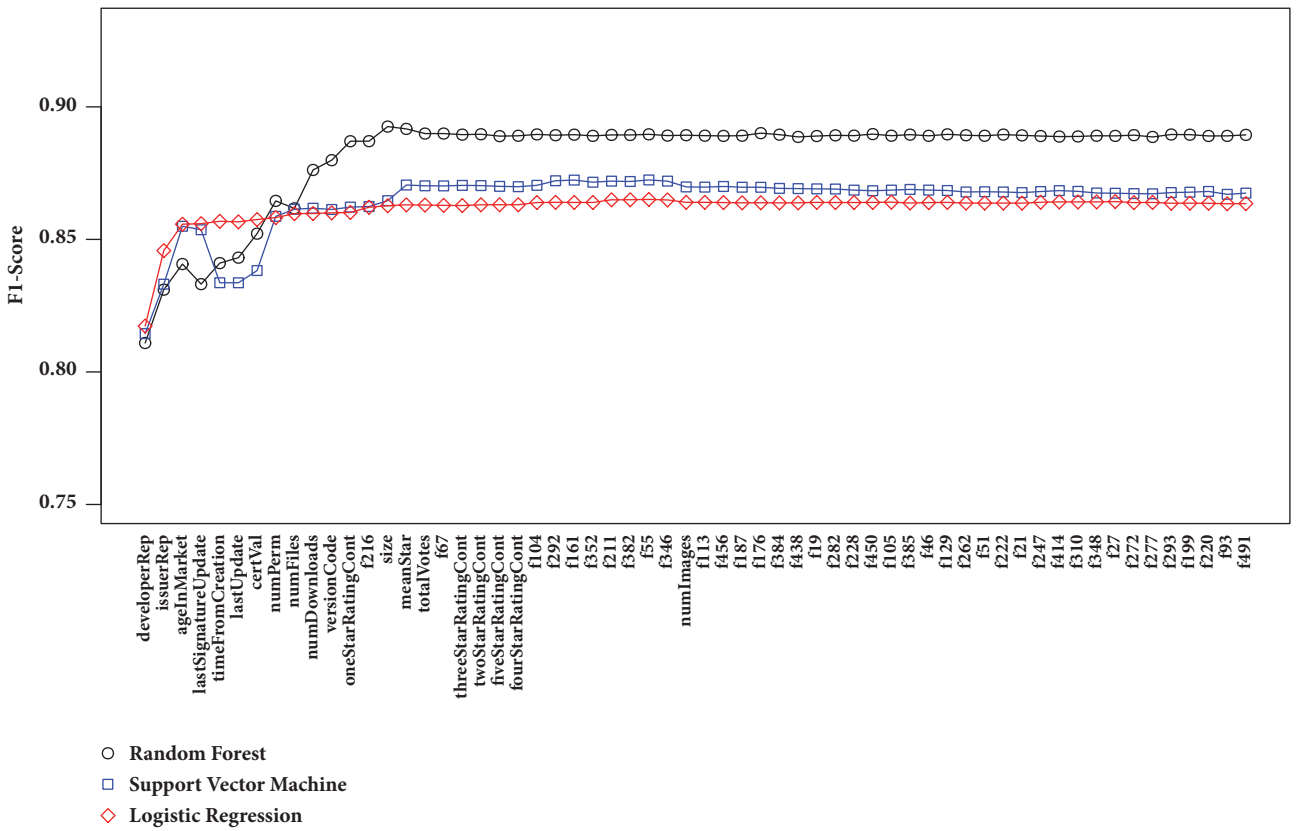
In addition, it is worth remarking that developerRep alone achieves an F1-score above 0.8, showing that this metric alone is more powerful than any other, including permissions.

5.3. Malware Detection Model. We perform a full benchmark test on the 9 composed datasets using only their top-15 features, namely, developerRep, issuerRep, ageInMarket, lastSignatureUpdate, timeForCreation, lastUpdate, certVal, numPerm, numFiles, numDownloads, versionCode, oneStarRatingCont, f216, size, and meanStar. As a result, Table 4 shows the training/test values of F1-score, precision, and recall metrics for each dataset and the three models under study (LR, SVM, and RF).

The results show that algorithms achieve similar results, slightly better in the case of RF. Second, it might be observed that general performance improves as the percentage of malware samples increases, showing best results when malware accounts for 50% of the applications. Actually, in the 2%-malware case, the difference between train and test error suggests that the algorithms are overfitting the data. Finally, the algorithms perform best at identifying those malware applications tagged by several AV engines. Clearly, when the algorithms are trained with malware applications tagged by two engines or more, they reach up to 0.87 F1-score in the test set (bottom line in the table), thus providing a high-level prediction confidence.



(a) Features sorted by importance



(b) Performance of classifiers with different number of features

Figure 4: Experiment results for feature selection.

Table 4: Full benchmark test with top-15 predictors.

Malware	NumDetectors	F1-score	Precision	Recall
Logistic Regression (train/test)				
2%	1	0.82/0.1	0.76/0.06	0.89/0.24
25%	1	0.89/0.57	0.87/0.46	0.91/0.73
50%	1	0.93/0.79	0.94/0.82	0.93/0.77
2%	2	0.8/0.18	0.74/0.12	0.88/0.33
25%	2	0.9/0.68	0.89/0.59	0.9/0.79
50%	2	0.94/0.82	0.95/0.78	0.93/0.86
2%	4	0.81/0.27	0.75/0.19	0.89/0.47
25%	4	0.91/0.73	0.9/0.65	0.91/0.83
50%	4	0.95/0.84	0.97/0.79	0.94/0.89
Support Vector Machine (train/test)				
2%	1	0.85/0.08	0.76/0.05	0.96/0.23
25%	1	0.93/0.68	0.92/0.69	0.93/0.67
50%	1	0.96/0.82	0.96/0.87	0.95/0.77
2%	2	0.82/0.16	0.72/0.1	0.95/0.35
25%	2	0.93/0.73	0.93/0.7	0.93/0.76
50%	2	0.96/0.84	0.97/0.9	0.94/0.8
2%	4	0.81/0.26	0.7/0.17	0.97/0.54
25%	4	0.94/0.77	0.94/0.72	0.93/0.83
50%	4	0.96/0.87	0.98/0.89	0.95/0.84
Random Forest (train/test)				
2%	1	0.99/0.12	0.99/0.07	0.99/0.33
25%	1	0.99/0.73	0.99/0.7	0.99/0.77
50%	1	0.99/0.84	0.99/0.88	0.99/0.8
2%	2	0.99/0.22	0.99/0.15	0.99/0.46
25%	2	0.99/0.77	0.99/0.73	0.99/0.83
50%	2	0.99/0.87	0.99/0.89	0.99/0.86
2%	4	0.99/0.32	0.99/0.22	0.99/0.59
25%	4	0.99/0.81	0.99/0.76	0.99/0.87
50%	4	0.99/0.89	0.99/0.88	0.99/0.9

Furthermore, it can be observed that the performance of metadata with respect to permissions only is highly superior, reaching larger F1-score values and using almost no permission hash in the process.

5.4. Robustness of the Model. The reader must note that malware developers, after reading this article, may decide to use different email accounts and certificates to evade this detection mechanism. However, the malwarish behaviour of applications is fingerprinted in several features redundantly, not only in the reputations. Truly, considering more than 13–15 features is completely unnecessary, as no extra predictive power is gained by adding new features (as shown in Figure 4); despite that, no less features should be selected, since most of them are redundant for performance, ensuring a certain degree of robustness for the model. This is very important, specially for cases when some features are corrupted or unavailable (i.e., the developer has changed accounts).

To show this, Table 5 shows the F1-score results of rerunning the RF algorithm to different subsets of features. Essentially, the first column shows the same train/test F1-score values as in Table 4 since both use the same top-15 features. The second column shows the F1-values when training and testing with features from 3 to 17 of Figure 4 (i.e., top-15 without developerRep and issuerRep). In this case, the F1-score value is slightly worse than before, but still the algorithm is able to classify malware accurately. Similarly using features 5-19 introduces a small decrease in F1-score, but still good performance is achieved. F1-score quickly drops when using the features from position 7 onwards in the ranking.

5.5. Performance and Impact of This Approach. The proposed methodology can be implemented as an early detection system that analyses metadata when a new application is submitted to an Android market. In this light, any delay introduced by this system in the market submission process could seriously impact the number of applications uploaded, as users are typically time-aware and not very keen on waiting too much. Thus, application analysis time is a key indicator for the success of the approach.

In our case, we have conducted an experiment to measure the time taken to build and test the models along with

Table 5: F1-score value of Random Forests with different feature sets.

F1-score Random Forest (train/test)							
NDet	1-15 feats.	3-17 feats.	5-19 feats	7-21 feats	9-23 feats.	11-25 feats	13-27 feats
1 AV	0.99/0.84	0.99/0.86	0.99/0.84	0.99/0.74	0.96/0.72	0.88/0.67	0.80/0.64
2 AV	0.99/0.87	0.99/0.87	0.99/0.86	0.99/0.79	0.96/0.75	0.88/0.71	0.75/0.66
4 AV	0.99/0.89	0.99/0.88	0.99/0.87	0.99/0.80	0.96/0.77	0.89/0.73	0.77/0.69

querying the model for a new app. The following numbers summarise our results in an Intel Xeon E5-2630 server with 24 cores and 190 GB of RAM memory:

(i) **Logistic Regression**: *Query*: 0.1 μs; *Building model (train+test)*: 46 ms; *Building model (train)* with Hyperparameter tuning (validation): 2 m 6 s,

(ii) **Support Vector Machines**: *Query*: 16 ms; *Building model (train)*: 2.4 s; *Building Model (train)* with Hyperparameter tuning: 8h 20m,

(iii) **Random Forest**: *Query*: 32 μs; *Building model (train+test)*: 3.5 s; *Building Model (train)* with Hyperparameter tuning: 47 m 32 s.

6. Summary and Discussion

In summary, this work has shown that Google Play metadata provides valuable information to detect Android malware applications, reaching F1-score values near 0.9, for example, when feeding metadata to a Random Forest. In particular, it has been shown that using no more than 15 features, malware applications can be accurately identified.

Furthermore, this work has also shown that inherent features, in particular application permissions, offer moderate prediction power (AUC-ROC about 0.7) compared to other metadata, such as the developer's reputation (percentage of malware applications uploaded by the same developer in the past) or certificate issuer reputation. This allows constructing efficient classification models for the early detection of malware applications uploaded at an Android market, as a prior step to more sophisticated techniques, namely, code inspection or sandboxing.

The results of this work show that metadata can be used as a simple static predictor for malware, specially suited to analyse at once large amounts of Android applications. This way, any application submitted to a market can be analysed to determine whether it has to be further inspected or can be directly uploaded. In addition, it is also possible to develop an in-device system which informs users about the appearance of each application and the risk of installing them in the device beforehand.

Furthermore, this methodology can be applied to other application markets like Aptoide or Amazon market, as they contain most of the metadata fields in Table 3, or equivalent ones that can be mapped to them.

In a nutshell, the contributions of this work are the following:

(i) We evaluated the capabilities of permission-based detection approaches and their limitations by means of the hashing trick as feature reduction technique.

(ii) We showed that inherent application features, such as the developer's reputation (percentage of malware applications uploaded by the same developer in the past) or certificate issuer's reputation, offer very good performance for detecting Android malware.

(iii) We proposed a model for Android malware detection based on metadata and machine learning techniques capable of detecting most Android threats, which can be leveraged both at market level and in-device application analysis.

(iv) We evaluated our proposed model over different benchmarking tests for performance and robustness of the algorithm.

Data Availability

The data used in this paper is property of Telefónica Identity & Privacy and, for strategic reasons, it cannot be disclosed. Nevertheless, the data has been collected from the information of publicly available applications in Google Play.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

The authors would like to acknowledge the support of the Spanish project TEXEO (Grant no. TEC2016-80339-R) and the EU-funded H2020 TYPES project (Grant no. H2020-653449). Additionally, Ignacio Martín would like to acknowledge the support of the Spanish Education Ministry for his FPU grant (Grant no. FPU15/03518) which supports his position at UC3M.

References

[1] Y. Zhou and X. Jiang, "Dissecting android malware: characterization and evolution," in *Proceedings of the 33rd IEEE Symposium on Security and Privacy*, pp. 95–109, San Francisco, Calif, USA, May 2012.

[2] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Proceedings of the NDSS Symposium Network and Distributed System Security*, 2014.

[3] S. Y. Yerima, S. Sezer, and G. McWilliams, "Analysis of bayesian classification-based approaches for android malware detection," *IET Information Security*, vol. 8, no. 1, pp. 25–36, 2014.

[4] J. Sahs and L. Khan, "A machine learning approach to android malware detection," in *Proceedings of the 2012 European Intelligence and Security Informatics Conference, EISIC '12*, pp. 141–147, August 2012.

[5] N. Peiravian and X. Zhu, "Machine learning for Android malware detection using permission and API calls," in *Proceedings of the 25th IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2013*, pp. 300–305, USA, November 2013.

[6] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," *Computers & Security*, vol. 65, pp. 121–134, 2017, http://www.sciencedirect.com/science/article/pii/S0167404816301601.

[7] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Information Security*, vol. 9, no. 6, pp. 313–320, 2015.

[8] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. Álvarez, "PUMA: Permission Usage to Detect Malware in Android," in *International Joint Conference CISIS'12-ICEUTE'12-SOCO'12 Special Sessions*, vol. 189 of *Advances in Intelligent Systems and Computing*, pp. 289–298, Springer, Berlin, Germany, 2013.

[9] H.-S. Ham and M.-J. Choi, "Analysis of Android malware detection performance using machine learning classifiers," in *Proceedings of the 2013 International Conference on Information and Communication Technology Convergence, ICTC 2013*, pp. 490–495, October 2013.

[10] K. O. Elish, X. Shu, D. D. Yao, B. G. Ryder, and X. Jiang, "Profiling user-trigger dependence for android malware detection," *Computers & Security*, vol. 49, pp. 255–273, 2015, http://www.sciencedirect.com/science/article/pii/S0167404814001631.

[11] Z. Wang, C. Li, Z. Yuan, Y. Guan, and Y. Xue, "DroidChain: A novel Android malware detection method based on behavior chains," *Pervasive and Mobile Computing*, vol. 32, pp. 3–14, 2016.

[12] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[13] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: scalable and accurate zero-day android malware detection," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, pp. 281–294, June 2012.

[14] Y. Aafer, W. Du, and H. Yin, "DroidAPIMiner: mining API-level features for robust malware detection in Android," in *Security and Privacy in Communication Networks*, pp. 86–103, Springer, 2013.

[15] K. Chen, P. Wang, Y. Lee et al., "Finding unknown malice in 10 seconds: Mass vetting for new threats at the google-play scale," in *Proceedings of the 24th USENIX Security Symposium (USENIX Security '15)*, pp. 659–674, 2015.

[16] B. Baskaran and A. Ralescu, "A study of android malware detection techniques and machine learning," 2016.

[17] Z. Aung and W. Zaw, "Permission-based Android malware detection," *Internation Jouranl of Scientific and Technology Research*, vol. 2, no. 3, pp. 228–234, 2013.

[18] D. Barrera, H. G. Kayacik, P. C. Van Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to android," in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pp. 73–84, ACM, October 2010.

[19] R. Johnson, Z. Wang, C. Gagnon, and A. Stavrou, "Analysis of android applications' permissions," in *Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability Companion, SERE-C 2012*, pp. 45-46, IEEE, 2012.

[20] M. Z. Mas'ud, S. Sahib, M. F. Abdollah, S. R. Selamat, and R. Yusof, "Analysis of features selection and machine learning classifier in android malware detection," in *Proceedings of the 5th International Conference on Information Science and Applications, ICISA '14*, pp. 1–5, IEEE, May 2014.

[21] Meet virustotal droidy, our new android sandbox, http://blog.virustotal.com/2018/04/meet-virustotal-droidy-our-new-android.html, accessed: June 2018.

[22] S. Zonouz, A. Houmansadr, R. Berthier, N. Borisov, and W. Sanders, "Secloud: A cloud-based comprehensive and lightweight security solution for smartphones," *Computers & Security*, vol. 37, pp. 215–227, 2013, http://www.sciencedirect.com/science/article/pii/S0167404813000310.

[23] K. Shaerpour, A. Dehghantanha, and R. Mahmod, "Trends in Android Malware Detection," *Journal of Digital Forensics, Security and Law*, vol. 8, no. 3, pp. 21–40, 2013.

[24] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM '11) Held in Association with the 18th ACM Conference on Computer and Communications Security (CCS '11)*, pp. 15–25, October 2011.

[25] D. Venugopal and G. Hu, "Efficient signature based malware detection on mobile devices," *Mobile Information Systems*, vol. 4, no. 1, pp. 33–49, 2008.

[26] M. Zheng, M. Sun, and J. C. S. Lui, "Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware," in *Proceedings of the 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom '13)*, pp. 163–171, July 2013.

[27] S. N. Hanumanthegowda, *Automated machine learning-based detection of malicious Android applications using Google Play Metadata [Master, thesis]*, Northeastern University, Boston, Mass, USA, 2013.

[28] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning ICML '09*, pp. 1113–1120, ACM, New York, NY, USA, June 2009.

[29] W. B. Tesfay, T. Booth, and K. Andersson, "Reputation based security model for android applications," in *Proceedings of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, TrustCom-2012*, pp. 896–901, June 2012.

[30] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*, vol. 103 of *Springer Texts in Statistics*, Springer, New York, NY, USA, 2013.

[31] A. Aswini and P. Vinod, "Droid permission miner: Mining prominent permissions for android malware analysis," in *Proceedings of the 2014 Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT '14)*, pp. 81–86, 2014.