Accepted Manuscript

Detecting android malicious apps and categorizing benign apps with ensemble of classifiers

Wei Wang, Yuanyuan Li, Xing Wang, Jiqiang Liu, Xiangliang Zhang

PII: S0167-739X(17)30074-2
DOI: http://dx.doi.org/10.1016/j.future.2017.01.019
Reference: FUTURE 3301

To appear in: *Future Generation Computer Systems*

Received date: 1 August 2016
Revised date: 29 November 2016
Accepted date: 15 January 2017

Please cite this article as: W. Wang, Y. Li, X. Wang, J. Liu, X. Zhang, Detecting android malicious apps and categorizing benign apps with ensemble of classifiers, *Future Generation Computer Systems* (2017), http://dx.doi.org/10.1016/j.future.2017.01.019

Highlights (for review)

- First work to provide a complete solution for automated categorization of apps
- Extract 2,374,340 features from each APK file
- Use ensemble of multiple classifiers to improve the detection accuracy
- Use large data set containing 107,327 benign apps and 8701 malapps for testing
- Employ five classifiers and compare the results of each classifier
- Reach detection accuracy as 99.39% and categorization accuracy as 82.93%

Detecting Android Malicious Apps and Categorizing Benign Apps with Ensemble of Classifiers

Wei Wang[a], Yuanyuan Li[a], Xing Wang[a], Jiqiang Liu[a], Xiangliang Zhang[b]

{wangwei1, liyuanyuan, spark, jqliu}@bjtu.edu.cn
xiangliang.zhang@kaust.edu.sa

[a] School of Computer and Information Technology, Beijing Jiaotong University, 3 Shangyuancun, Beijing 100044, China

[b] Division of Computer, Electrical and Mathematical Sciences & Engineering, King Abdullah University of Science and Technology (KAUST), Saudi Arabia

Android platform has dominated the markets of smart mobile devices in recent years. The number of Android applications (apps) has seen a massive surge. Unsurprisingly, Android platform has also become the primary target of attackers. The management of the explosively expansive app markets has thus become an important issue. On the one hand, it requires effectively detecting malicious applications (malapps) in order to keep the malapps out of the app market. On the other hand, it needs to automatically categorize a big number of benign apps so as to ease the management, such as correcting an app's category falsely designated by the app developer. In this work, we propose a framework to effectively and efficiently manage a big app market in terms of detecting malapps and categorizing benign apps. We extract 11 types of static features from each app to characterize the behaviors of the app, and employ the ensemble of multiple classifiers, namely, Support Vector Machine (SVM), K-Nearest Neighbor (K-NN), Naive Bayes (NB), Classification and Regression Tree (CART) and Random Forest (RF), to detect malapps and to categorize benign apps. An alarm will be triggered if an app is identified as malicious. Otherwise, the benign app will be identified as a specific category. We evaluate the framework on a large app set consisting of 107,327 benign apps as well as 8,701 malapps. The experimental results show that our method achieves the accuracy of 99.39% in the detection of malapps and achieves the best accuracy of 82.93% in the categorization of benign apps.

Android security; malware detection; intrusion detection; classification; ensemble learning; static analysis

1 Introduction

Mobile devices have become increasingly popular in both personal and business use. Android has dominated the markets of mobile devices. The management of the explosively expanding application (app) markets has thus become an emerging and crucial issue that involves the detection of malicious applications (malapps) as well as the categorization of benign apps. On the one hand, Android platform has seen a massive surge in malapps. The increasingly sophisticated Android malpps are motivated to evade the detection systems. Therefore, the management of Android app markets calls for effective methods for detecting malapps. On the other hand, it requires to automatically categorize a large number of apps so as to ease the management. An app market usually categorizes an app with the category specified by the app's developers or by analysing the descriptions that the app's developers provide. This process can easily be manipulated by a malapp's developer to evade the detection. Furthermore, as the number of apps explosively increases, it is crucial to automatically categorize apps quickly and accurately so as to improve the efficiency of management.

In this work, we propose a novel framework to effectively and efficiently manage a large number of apps in an app market in terms of detecting malapps and catego-

rizing benign apps. We extract a large number of features from each Android Application Package (APK) file and use these features to characterize the behaviors of the app. In general, the behaviors of malapps discriminate from those of benign apps. For example, a malicious app may request a permission that it functionally does not need. The behaviors of benign apps in each category also discriminate from those in other categories. From each app, in total we extract 2,374,340 features that fall into 11 types. In order to improve the efficiency of the detection and of the categorization, we employ Support Vector Machine (SVM) to rank the features according to their importance to the detection. Finally, we choose the top ranked 34,630 features for the detection of malapps and the categorization of benign apps. We then employ the ensemble of five classifiers including SVM, Random Forest (RF), K-Nearest Neighbors (K-NN), Classification and Regression Tree (CART), and Naive Bayes (NB), to detect malapps and to categorize benign apps.

Through the extensive analysis, we find that the game category has some specific characteristics. The game category is very large, in which there are many subcategories. Some of these subcategories have a bigger number of samples than other subcategories. Besides, the behaviors of game apps are quite different from those of non-game apps. For example, apps in game category usually generate more network traffic and have much in-app purchases. In order to explorer the differences between game and non-game categories, as shown in Fig. 1, if an app is identified as benign, our framework will first predict whether it is a game, and then specify the app's game category. Otherwise, it will be predicted as a specific non-game category. We evaluate our framework on a large app set consisting of 18,866 game apps, 88,461 non-game apps as well as 8,701 malapps. In the experiments of malapp detection, our ensemble method achieves the detection accuracy as 99.39%. The experimental results also show that our ensemble method achieves the accuracy of 66.23% in categorizing game apps and the accuracy of 82.93% in categorizing non-game apps.

We make the following contributions in this paper:

- To the best of our knowledge, this is the first work to provide a complete solution for the automated categorization of benign apps with ensemble of multiple classifiers. Our ensemble method outperforms the five base classifiers in overall categorization of benign apps, achieving the best accuracy of 82.93%.
- Complementing to Drebin [1] in which the feature set contains 8 types, we extract as many as 2,374,340 features that fall into 11 types from each APK file. In order to improve the efficiency of the detection and categorization, we rank the 2,374,340 features with SVM according to their importance to the detection and choose the top ranked 34,630 static features in our work for the detection and categorization.
- We conduct a series of comprehensive experiments to evaluate the proposed framework using a very large app set consisting of 107,327 benign apps and 8,701 malapps.
- We employ five classifiers in our framework, and compare and analyze the results of each classifier. The result is determined by using the ensemble of five classifiers to balance the performance of all classifiers.

The remainder of this paper is organized as follows. Section 2 reviews the background and related work. Section 3 describes the methods. Section 4 presents the evaluation results and Section 5 concludes this paper.

2 Background and Related Work

The security of computer systems and networks has been a widely studied topic as attacks against systems or network infrastructures are currently major threats. Many methods have been proposed to ensure system security [1-3] or network security [4-9].

The motivation of our work is to ensure the security of Android apps. Existing work on the detection of malapps mainly focuses on the analysis of static features [10-15] or dynamic features [16-18]. As Android permissions limit an app with Application Program Interface (API) calls [19], the requested permissions can characterize partial behaviors of apps. Pandita et al. [12] proposed WHYPER that employed natural language processing technology to explain why apps need certain permissions. In our previous work [21], we ranked the permissions according to their risks and systematically studied how well Android permissions perform in the detection of malapps. Barrera et al. [22] surveyed 1100 most popular apps and found that these apps only required a small portion of the permissions. They also found that the relationship between app categories and permissions is not very close. Felt and Chia et al. [23, 24] studied Android apps and listed the permissions that were most commonly used in apps. Enck et al. [25] developed a tool to check whether an app is malicious by matching its requested permissions with pre-defined permission lists. Zhou et al. [26] proposed a permission-based behavioral footprinting scheme to detect new samples of known Android malware families and applied a heuristics-based filtering scheme to identify certain inherent behaviors of unknown malicious families.

Shabtai et al. [27] studied the techniques of static analysis to analyze Android source code. They also applied machine learning techniques to categorize games and tools with static features extracted from Android

apps. La Polla et al. [28] provided an overview of related work on security of mobile devices. Nath et al. [29] compared various machine learning techniques used for analyzing Android malwares. Lindorfer et al. [30] presented MARVIN, a system that leverages machine learning techniques to assess the risk associated with unknown Android apps in the form of a risk score. Peiravian et al. [31] detected malapps with permissions and API calls. Pirscoveanu et al. [32] developed a distributed malware testing environment. Amin et al. [33] investigated the natures and identities of malapps and proposed network-based and system call based detection approaches. Apvrille et al. [34] extracted code-level features and classified unknown apps with Alligator that combines several classification algorithms. In our previous work, we developed a static analysis tool called SDFDroid [35] that identifies the sensors' types and generates the sensor data propagation paths in each app to provide an up-to-date overview of sensor usage patterns. The sensor usage patterns are then used to identify whether an app has potential threat. In order to detect zero-day malapps, we developed a system called Anomadroid (anomaly Android malapp detection system) [36] that profiles the normal behaviors of apps based on only benign samples. Any app whose behaviors unacceptably deviate from the normal profile is identified as malicious. We also presented a system called Alde [37] to analyze privacy risk of analytics libraries in the Android ecosystem.

Although there exists research effort on the detection of malapps, no much work has focused on the categorization of apps. In the most related work, Shabtai et al. [27] classified two categories of apps including tools and games with 3 types of features extracted from apps. In contrast, in this work we extract 11 types of features to categorize each app into 24 categories designated by an app market with ensemble of multiple classifiers.

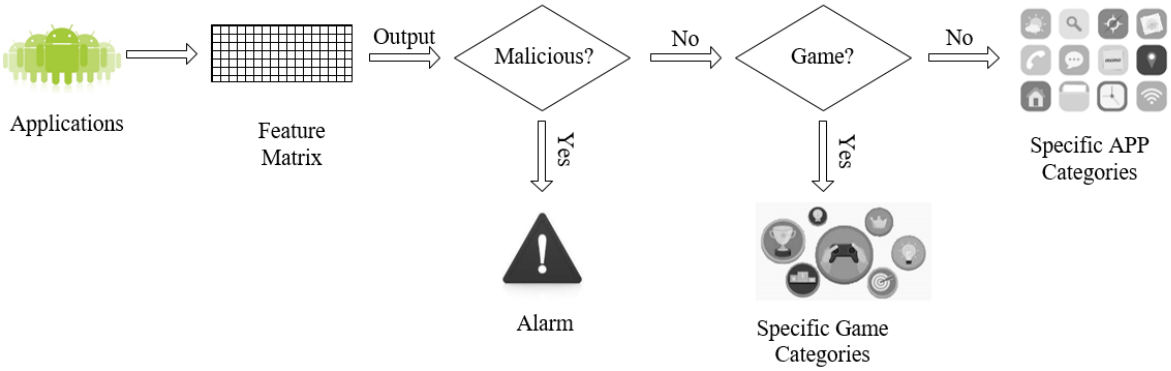

Figure 1: The overview of our method

3 Method

3.1 Overview

We propose a framework to manage a big app market in terms of detecting malapps and categorizing benign apps. As described in Figure 1, after an app is fed into the framework, a big number of features will be extracted from the APK file. An alarm is triggered if it is identified as malicious with ensemble of multiple classifiers. Otherwise, it is further predicted as whether a game app. If it is predicted as a game, our framework will specify its game category. Otherwise it will be given a specific non-game category.

3.2 Feature Extraction

The extraction and selection of features are always important in detecting malapps and in categorizing benign apps. In this work, we extract static features from apps with static analysis. Static analysis techniques refer to analyzing the code features, such as the grammar, lexicon, data flow and control flow. Compared to dynamic analysis in which features are extracted from the system while the app is running, applying static analysis consumes much less resources and time, and does not require to execute the app. As we are motivated to manage a big app market, in this work we focus on static analysis to ensure the detection efficiency.

With static analysis, we extract 2,374,340 features from each app. The number of features is too large to be efficiently processed. Therefore, we use SVM algorithm to sort the weight of each feature according to its importance to the detection. In detail, we apply linear SVM for malapp detection and achieve an SVM model in which each feature is given a weight determined by how effective it is in distinguishing the benign apps and

malapps. The features whose weights are positive are treated as "malicious features", and the negative ones are treated as "benign features". Some weights equal to zero, indicating that they do not contribute to the categorization and thus can be removed in the analysis. We rank the features according to the absolute values of weights, and finally use the top ranked 34,630 features to perform the detection of malapps and the categorization of benign apps. The feature sets we used are described in Table 1.

TABLE 1. Descriptions of Feature Sets

	Feature Type	Number of Features
1	Request Permissions	96
2	Filtered Intents	126
3	Restricted API calls	34,188
4	Code-related Information	5
5	Used Permission	96
6	Hardware Features	41
7	Suspicious API calls	78

3.3 The Algorithms for the Detection and Categorization

(1) Support Vector Machine (SVM)

SVM [38] is a supervised machine learning method for classification and regression analysis. The principle of SVM is to find the best hyperplane to separate the data into two parts. A SVM model consists of the samples represented as points in space. The samples of the different categories are divided by hyperplane. This hyperplane always maximizes the margin between those two regions or classes. The margin is defined by the farthest distance between the samples of the two classes and computed based on the distances between the closest samples of both classes, which are called supporting vectors. Test samples are then mapped into the same space. Based on which sides of the hyperplane they fall on, test samples are predicted to belong to the corresponding categories.

In the experiments of the multi-class app categorization, we use multi-stage SVM. It is similar to the theory described above. The only difference is that the multi-stage SVM creates multiple hyperplanes. Each hyperplane will separate the samples into one specific category and the residual category.

(2) Classification And Regression Tree (CART)

CART is a method based on the Gini index. It usually uses a top-down approach when CART constructs a decision tree. Decision tree [39] is a categorization model that recursively partitions the training data into a tree structure.

In the experiments, we first put all the training samples at the root node. We then search the best partition of the root node so that the Gini impurity can reduce to minimum. Gini impurity represents the possibility that a randomly selected sample is classified into the wrong subset. When all the samples of a node belong to one class, Gini impurity equals to zero. We use the best partition to divide root node into two parts, each of which is seen as a new node. This process is then repeated on the new nodes.

(3) Random Forest (RF)

RF [40] algorithm itself is an ensemble classifier that constructs multiple decision trees. Each tree is learned independently on a randomly selected subset of training data. A subset for training each decision tree comes from randomly sampling for both features and apps. The output category is determined by the mode of categories that all of the trees output. In general, random forest is more suitable for multi-categorization.

(4) Naive Bayes (NB)

Naive Bayes is a probabilistic classifier based on Bayes theorem [41]. Given a test sample, we need to calculate the probabilities of the appearance of various categories under the condition of the appearance of a test sample. The sample belongs to the category whose probability is the largest.

(5) K-Nearest Neighbor (K-NN)

K-NN [42] algorithm is a non-parametric statistical methods for categorization and regression. It classifies a test sample by measuring the distance between the training samples and test sample. We need to choose *k* nearest samples and use majority voting to predict which category the sample belongs to.

(6) Ensemble of Classifiers

To exert the advantage of each algorithm and to further improve the accuracy of detection and categorization, we employ the ensemble of multiple classifiers with majority voting after obtaining the classification results of the five algorithms described above. The algorithm of the ensemble of multiple classifiers is shown in Figure 2. When a test app is given as input, each base classifier predicts its classification. All the five prediction results will then vote to generate a final prediction. Basically the ensemble of multiple classifiers with voting mechanism outperforms a base classifier in prediction.

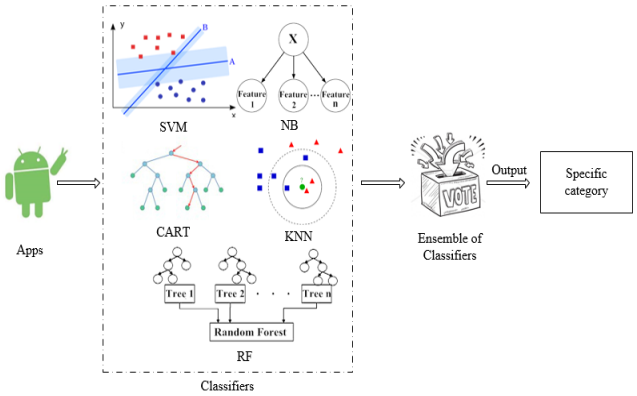

Figure 2. The ensemble of multiple classifiers

4 Evaluation

4.1 App Sets

We use a big app set consisting of benign apps and malapps to validate our framework. The benign apps were crawled from one of the biggest app markets in China called Anzhi [43] and malapps were collected in wild. In total, we collected 116,028 apps consisting of 107,327 benign apps and 8,701 malapps. We labeled the benign apps with VirusTotal [44]. If less than an anti-virus engine in VirusTotal recognizes the app as malicious, we then deem it as a benign app. We randomly select 90% of the samples to serve as the training data. The remaining 10% of the samples are used as the test data.

In the experiments of app categorization, we first detect whether an app is a game app, as game apps have distinct characteristics. If an app is detected as a game app, our framework will further identify its specific game category. Otherwise if it is predicted as a non-game app, our framework will identify its specific non-app category. The 107,327 benign apps are distributed into 24 app categories, 8 of which are game categories. As categorization methods for each market are not the same, we manually calibrate the app categories with the categorization information provided by Anzhi market. The categories of game apps and of non-game apps are shown in Table 2 and Table 3, respectively. Same with the experiments of malapp detection, we randomly select 90% of the samples to serve as the training data and the remaining 10% samples are used as the test data. The data we used in the experiments is described in Table 4. We transform the app samples into LIBSVM format. Each line describes an app sample. The first column indicates the class label, and the rest of the columns describe the features.

TABLE 2. Descriptions of Game Samples

	Categories	Number of Samples
1	G_ACTION	2,832
2	G_BRAIN_CARDS_AND_CASUAL	11,509
3	G_FLIGHT_GAMES	367
4	G_ONLINE_GAMES	390
5	G_RPG	1,164
6	G_SIMULATION	497
7	G_SPORTS_AND_RACING	1,307
8	G_STRATEGY	800

TABLE 3. Descriptions of Non-Game Samples

	Categories	Number of Samples
1	A_BOOKS_READER_AND_MAGAZINES	14,563
2	A_BROWSER	190
3	A_FINANCE	1,440
4	A_INPUT_METHOD	62
5	A_LIFE	21,674
6	A_MUSIC	1,995
7	A_NEWS	1,738
8	A_OFFICE_AND_BUSINESS	4,464
9	A_PHOTOGRAPHY_AND_BEAUTIFICATION	866
10	A_SECURITY	261
11	A_SHOPPING_AND_PAYMENT	2,605
12	A_SOCIAL_AND_COMMUNICATION	3,428
13	A_THEMES_AND_WALLPAPER	29,311
14	A_TOOLS	3,031
15	A_TRANSPORTATION	1,589
16	A_VIDEO	1,244

4.2 The Parameters of Training Models

In the evaluation, we use a computer with a quad-core CPU and memory of 64G. We employ scikit-learn [45] packages written in Python as the machine learning tool in the experiments, as it is accessible and reusable in various contexts.

For SVM, we choose the linear kernel function in all experiments. The parameter w_i, which means the weight of each category, is assigned as inversely proportional to the number of samples in each category. Another parameter C, the penalty parameter of the error, is set as 0.25.

The number of features is an important parameter for RF, as it is required to consider when looking for the best split. Another parameter is the number of decision trees. We balance the efficiency and accuracy through several experiments, and set both the number of decision trees and the number of features as 300.

Similarly, the number of features is an important parameter for CART. We set this parameter as default, as the accuracy is the best with the default value. We set K = 9 for K-NN. As for NB, we choose MultinomialNB model in our experiments.

TABLE 4. Overview of the Apps Used in the Experiments

	Number of Training samples	Number of Test Samples	Total number
Malapp Detection	104,425	11,603	116,028
Game app Detection	96,594	10,733	107,327
Game App categorization	16,979	1,887	18,866
Non-game App categorization	79,615	8,846	88,461

4.3 Experimental Results and Analysis

(1) Detecting Malapps

The detection results of malapps for each classifier are shown in Table 5. It is seen that the True Positive Rates (TPR) of SVM are the highest among the five classifiers, achieving 96.07%. The evaluation results show that the detection accuracy of each algorithm is quite similar except NB algorithm. It is also seen from Table 5 that the detection results of CART algorithm and RF algorithm are quite similar. However, in general, the detection results of RF algorithm are better than those of CART algorithm. RF outperforms the other four classifiers and its detection accuracy reaches 99.49%.

Our method achieves the accuracy of 99.39% in the detection of malapps after employing ensemble of the five classifiers with majority voting mechanism. In general, our method outperforms SVM, NB, K-NN and CART. Although our method does not excel RF, the detection performance between the two methods is comparable.

(2) Identifying Game or Non-Game Apps

If an app is predicted as benign, we will further identify whether the app belongs to the game categories or non-game categories. This process aims to distinguish between game apps and non-game apps, which is different from malapp detection. In the experiments, we consider this problem as a kind of multi-classification. Therefore, we only calculate the detection accuracy and detection rates of game or non-game apps.

TABLE 5. The Detection Results of Malapps with the Five Base Classifiers as well as with Ensemble of Classifiers

Classifier	Accuracy	TPR	FPR	Precision	Recall	F-measure
SVM	98.82%	96.07%	3.39%	92.79%	95.09%	93.93%
RF	99.49%	93.74%	0.07%	98.96%	93.74%	96.28%
NB	76.46%	90.92%	24.63%	21.73%	90.92%	35.08%
K-NN	97.95%	76.69%	0.45%	92.73%	76.69%	83.95%
CART	99.23%	95.83%	0.52%	93.31%	95.83%	94.55%
Ensemble	99.39%	93.25%	0.15%	97.94%	93.25%	95.54%

The overall detection accuracy and detection rates of game apps or non-game apps with the five classifiers and ensemble method are shown in Table 6. The overall detection accuracy of RF is the highest, reaching 97.22%. Our ensemble method achieves the best detection rate of game apps. Although the ensemble method is slightly worse than RF in terms of the detection rates of non-game apps and overall detection accuracy, its overall detection performance is more robust than all the five classifiers.

TABLE 6. The Detection Results for Game Apps and Non-Game Apps with the Five Base Classifiers as well as with Ensemble of Classifiers

Classifier	Detection Rate of Game apps	Detection Rate of Non-Game apps	Overall Detection Accuracy
SVM	92.43%	91.50%	91.66%
RF	88.82%	98.99%	97.22%
NB	58.60%	96.19%	89.68%
K-NN	75.91%	98.02%	94.19%
CART	88.06%	97.60%	95.95%
Ensemble	**93.10%**	97.57%	96.91%

(3) Categorizing Game Apps

If an app is predicted as game, we will then further identify which specific game category it belongs to. In total there are 8 categories for game apps. The overall categorization accuracy of game apps with five classifiers as well as with ensemble of classifiers are shown in Table 7.

TABLE 7. The Overall Categorization Results for Game Apps with the Five Base Classifiers as well as with the Ensemble of Classifiers

Classifier	SVM	RF	NB	K-NN	CART	Ensemble
Accuracy	53.86%	66.08%	55.27%	60.78%	58.19%	**66.23%**

It is seen from Table 7 that our ensemble method outperforms the five base classifiers in terms of the overall categorization accuracy for game apps.

We investigate the categorization accuracy of each category for game apps and the results are shown in Figure 3 and Table 8. The categorization accuracy of G_BRAIN_CARDS _AND_CASUAL is the best for all the classifiers. The categorization accuracies of G_FLIGHT _GAMES and G_STRATEGY are poor. Our ensemble method achieves the best categorization accuracy as 93.99% for G_BRAIN_CARDS_AND_CASUAL. In general, the ensemble method performs more robust than the five base classifiers, although its categorization accuracy for other 7 game categories is not the best.

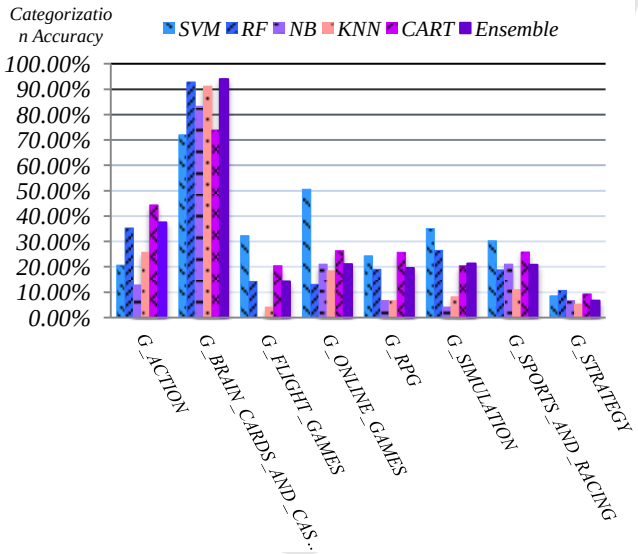

Figure 3. The Categorization Accuracy for 8 Game App Categories with the Five Base Classifiers and with Ensemble of Classifiers.

TABLE 8. The Categorization Accuracy for 8 Game App Categories with the Five Base Classifiers and with Ensemble of Classifiers

Classifier	SVM	RF	NB	K-NN	CART	Ensemble
G_ACTION	20.67%	35.25%	12.88%	25.76%	44.41%	37.51%
G_BRAIN_CARDS_AND_CASUAL	71.98%	92.86%	83.28%	91.20%	73.95%	**93.99%**
G_FLIGHT_GAMES	32.00%	14.29%	0.01%	4.08%	20.41%	14.29%
G_ONLINE_GAMES	50.59%	13.16%	21.05%	18.42%	26.32%	21.05%
G_RPG	24.37%	19.05%	6.67%	6.67%	25.71%	19.47%
G_SIMULATION	35.05%	26.53%	4.08%	8.16%	20.41%	21.33%
G_SPORTS_AND_RACING	30.29%	18.75%	21.09%	10.94%	25.78%	20.84%
G_STRATEGY	8.61%	10.67%	6.67%	5.33%	9.33%	6.67%

We analyze the app sets and find that G_BRAIN _CARDS_AND_CASUAL consists of puzzle & card games, which are very popular among users. The number of apps in this category reaches 11,509 that is much bigger than that of other game categories. The number of apps in G_FLIGHT_GAMES and G_ONLINE_GAMES are only 367 and 390, respectively. The number of apps in different game categories is clearly imbalanced. In addition, the game functions and operations of G_BRAIN_CARDS_AND_CASUAL are simpler than those of G_FLIGHT_GAMES and G_SIMULATION. The categorization results for G_BRAIN_CARDS_AND_CASUAL is thus the best for all the classifiers.

In general, the categorization accuracy for game apps is not high. The first reason is that the definitions of game categories are not very clear. For example, some games in G_FLIGHT_GAMES can also belong to

G_SPORTS_AND_RACING. Some other game apps belong to two categories or even to several categories. The second reason is that the functions and operations of all the games are always quite similar to each other. Both these two reasons result in low categorization accuracy.

(4) Categorizing Non-Game Apps

If an app is predicted as non-game, we will further identify which specific category it belongs to. In total, there are 16 categories for non-game apps. The overall categorization accuracies with five classifiers and with our ensemble method are shown in Table 9. It is clear that our ensemble method outperforms the five base classifiers, achieving the categorization accuracy as 82.93%.

TABLE 9. The Overall Categorization Accuracy for Non-Game Apps with the Five Base Classifiers and with the Ensemble of Classifiers

	SVM	RF	NB	K-NN	CART	Ensemble
Accuracy	80.85%	82.88%	65.14%	77.56%	79.88%	**82.93%**

Table 10 shows the categorization accuracy for each non-game category with the five base classifiers and with our ensemble method. In general, our ensemble method is more robust than the five classifiers in categorization. It is seen from Table 10 that our ensemble method achieves the best categorization accuracy for A_BOOKS_READER _AND_MAGAZINES, A_LIFE, A_SOCIAL_AND_COMMUNICATION and A_THEMES_AND_WALLPAPER. Apps in A_BOOKS_READER_AND_ MAGAZINES or A_ THEMES_AND_WALLPAPER are popular, and therefore the number of apps in these categories is very large. In addition, the app functions of these two categories are simple and similar, and thus the features extracted are very similar, too. For example, the features extracted from apps in A_BOOKS_READER _AND_MAGAZINES are mostly like "hardware_ feature::android.hardware.touchscreen", "permission::android. permission.INTERNE" and "used_ permission::ACCESS_NETWORK_STATE". The features extracted from apps in A_THEMES_AND_WALLPAPER are mostly like " hardware_feature::android.hardware.touchscreen" and " file_type_number::Dalvik". These features are very common in most apps in these categories, and this explains why the categorization accuracy for these categories are very high.

However, the categorization accuracy of all the classifiers is very poor for certain categories, such as A_BROWSER and A_SECURITY that represent browsers and mobile security tools. Apps of these two categories have various functions. Therefore, features extracted from these apps are diversely distributed. Meanwhile, the number of apps in these two categories is very small. These may explain why certain categories like A_BROWSER and A_SECURITY are difficult to categorize.

TABLE 10. The Categorization Accuracy for 16 Non-Game App Categories with the Five Base Classifiers and with the Ensemble of Classifiers

Category	SVM	RF	NB	K-NN	CART	Ensemble
A_BOOKS_READER_AND_MAGAZINES	98.26%	99.66%	90.21%	99.80%	98.93%	**99.87%**
A_BROWSER	41.18%	11.76%	0.01%	0.01%	17.65%	17.65%
A_FINANCE	30.15%	18.79%	4.70%	8.05%	23.49%	18.79%
A_INPUT_METHOD	60.00%	20.00%	0.00%	20.00%	20.00%	20.00%
A_LIFE	88.70%	92.50%	51.06%	87.13%	79.81%	**93.66%**
A_MUSIC	77.27%	66.16%	32.83%	56.06%	70.71%	70.71%
A_NEWS	33.89%	44.44%	27.78%	32.78%	43.33%	39.44%
A_OFFICE_AND_BUSINESS	36.51%	52.09%	27.44%	39.77%	54.88%	50.47%
A_PHOTOGRAPHY_AND_BEAUTIFICATION	34.83%	31.46%	3.37%	15.73%	32.58%	32.58%

A_SECURITY	47.24 %	10.34 %	0.00 %	0.00%	10.34%	10.34%
A_SHOPPING_AND_PAYMENT	38.78 %	33.95 %	15.50 %	25.09%	37.64%	28.78%
A_SOCIAL_AND_COMMUNICATION	36.48 %	35.22 %	48.43 %	23.90%	37.42%	**37.42%**
A_THEMES_AND_WALLPAPER	97.60 %	98.53 %	93.47 %	97.40%	98.36%	**98.67%**
A_TOOLS	39.10 %	44.23 %	25.96 %	12.50%	36.86%	40.38%
A_TRANSPORTATION	35.71 %	40.71 %	34.29 %	35.00%	43.57%	39.29%
A_VIDEO	53.03 %	53.03 %	9.09 %	31.82%	51.52%	51.52%

5 Conclusion

In this work, we propose a framework to detect malapps and to categorize benign apps with ensemble of five classifiers, in the aim to ease the management of Android app markets. Given an app, our framework firstly extracts a large number of features from the app, and use these features to detect whether it is malicious or not. An alarm is triggered if it is identified as malicious. Otherwise, it will be categorized as a specific category. We employ ensemble of five classifiers, namely, SVM, K-NN, NB, CART and RF with majority voting for the detection of malapps and the categorization of benign apps. The experimental results show that our ensemble method is more robust than the five base classifiers in the detection and categorization. In the experiments of malapp detection, our ensemble method achieves the detection accuracy as 99.39%. The extensive experimental results also show that our framework achieves the best overall categorization accuracy with the accuracy of 66.23% in categorization of game apps and the accuracy of 82.93% in categorization of non-game apps.

In future work, we plan to explore more informative features to better characterize the behaviors of apps to improve the performance of the detection of malapps and categorization of benign apps. Designing more effective ensemble algorithms is also being investigated.

6 Acknowledgement

The work reported in this paper was supported in part by the Scientific Research Foundation through the Returned Overseas Chinese Scholars, Ministry of Education of China, under Grant K14C300020, in part by Shanghai Key Laboratory of Integrated Administration Technologies for Information Security, under Grant AGK2015002, in part by ZTE Corporation foundation, and in part by National Natural Science Foundation of China, under Grant 61672092.

Reference

[1]. W. Wang, X. Zhang, S. Gombault, "Constructing attribute weights from computer audit data for effective intrusion detection. Journal of Systems and Software", 2009, 82(12): 1974-1981.

[2]. X. Guan, W. Wang, X. Zhang, "Fast intrusion detection based on a non-negative matrix factorization model". Journal of Network and Computer Applications, 2009, 32(1): 31-44.

[3]. W. Wang, X. Guan, X. Zhang, L. Yang, "Profiling program behavior for anomaly intrusion detection based on the transition and frequency property of computer audit data". Computers & security, 2006, 25(7): 539-550.

[4]. X. Zhang, C. Furtlehner, C. Germain-Renaud, M. Sebag, "Data stream clustering with affinity propagation". IEEE Transactions on Knowledge and Data Engineering, 2014, 26(7): 1644-1656.

[5]. W. Wang, X. Guan, X. Zhang, "Processing of massive audit data streams for real-time anomaly intrusion detection". Computer communications, 2008, 31(1): 58-72.

[6]. W. Wang, J. Liu, G. Pitsilis, J. Liu, "Abstracting massive data for lightweight intrusion detection in computer networks". Information Sciences (online first), 2016.

[7]. X. Zhang, T. Lee, G. Pitsilis, "Securing recommender systems against shilling attacks using social-based clustering". Journal of Computer Science and Technology, 2013, 28(4): 616-624.

[8]. W. Wang, T. Guyet, R. Quiniou, M. Cordier, F. Masseglia, X. Zhang, "Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks". Knowledge-Based Systems, 2014, 70: 103-117.

[9]. W. Wang, R. Battiti, "Identifying intrusions in computer networks with principal component analysis", First International Conference on Availability, Reliability and Security. IEEE, 2006: 1-8.

[10]. D. Arp, M. Spreitzenbarth, M. Hübner and H. Gascon, K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," Proceedings of NDSS. 2014.

[11]. H. Peng, C. Gates and B. Sarma, et al. "Using probabilistic generative models for ranking risks of android apps," Proceedings of the 2012 ACM conference on Computer and communications security, pp. 241-252, 2012.

[12]. B. P. Sarma, N. Li, C. Gates, R. Potharaju and C. Nita-Rotaru, "Android permissions: a perspective combining risks and benefits," Proceedings of the 17th ACM symposium on Access Control Models and Technologies, pp. 13-22, 2012,06.

[13]. R. Pandita, X. Xiao, W. Yang, W. Enck and T. Xie, "WHYPER: Towards Automating Risk Assessment of Mobile Applications," the 22nd USENIX Security Symposium (USENIX Security 13), pp. 527-542, 2013, 13.
[14]. W. Enck, D. Octeau, P. McDaniel and S. Chaudhuri, "A Study of Android Application Security," Proceedings of the 20th USENIX conference on Security, pp. 21-36, 2011.
[15]. M. Grace, Y. Zhou, Q. Zhang, S. Zou and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," Proceedings of the 10th international conference on Mobile systems, applications, and services. ACM, pp. 281-294, 2012.
[16]. L. Lu, Z. Li, Z. Wu, W. Lee and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities" Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 229-240, 2012.
[17]. W. Enck, P. Gilbert, B. Chun, et al. "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," Proceedings of OSDI'10 Proceedings of the 9th USENIX conference on Operating systems design and implementation, pp. 393-407, 2014.
[18]. P. Hornyack, S. Han, J. Jung, S. Schechter and D. Wetherall, "These aren't the droids you're looking for: retrofitting android to protect data from imperious applications," Proceedings of the 18th ACM conference on Computer and communications security, ACM, pp. 639-652, 2011.
[19]. V. Rastogi, Y. Chen and W. Enck. "AppsPlayground: automatic security analysis of smartphone applications," Proceedings of the third ACM conference on Data and application security and privacy. ACM, pp. 209-220, 2013.
[20]. M. Frank, B. Dong, A. P. Felt and D. Song, "Mining Permission Request Patterns from Android and Facebook Applications" Data Mining, IEEE 12th International Conference on, pp. 870-875, 2012, 10.
[21]. W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, X. Zhang, "Exploring Permission-induced Risk in Android Applications for Malicious Application Detection," Information Forensics and Security, IEEE Transactions on, pp.1869-1882, 2014,09.
[22]. 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," Proceedings of the 17th ACM conference on Computer and communications security. ACM, pp. 73-84, 2010,10.
[23]. P. H. Chia, Y. Yamamoto and N. Asokan, "Is this app safe?: a large scale study on application permissions and risk signals," Proceedings of the 21st international conference on World Wide Web. ACM, pp. 311-320, 2012, 02.
[24]. A. P. Felt, K. Greenwood and D. Wagner, "The effectiveness of application permissions," Proceedings of WebApps'11 Proceedings of the 2nd USENIX conference on Web application development, pp . 7-7, 2011, 06.
[25]. W. Enck, M. Ongtang and P. McDaniel, "On lightweight mobile phone application certification" Proceedings of the 16th ACM conference on Computer and communications security. ACM, pp. 235-245, 2009.
[26]. Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets" Proceedings of the 19th Network and Distributed System Security Symposium NDSS, 2012 .
[27]. A. Shabtai, Y. Fledel and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," Computational Intelligence and Security (CIS), 2010 International Conference on. IEEE, 329-333, 2010,12.
[28]. M. La Polla, F. Martinelli and D. Sgandurra, "A survey on security for mobile devices," Communications surveys & tutorials, IEEE, 15(1) pp. 446-471, 2013,03.
[29]. H. Nath, B. Mehtre, "Static Malware Analysis Using Machine Learning Methods," Communications in Computer and Information Science (CCIS2014) pp.440-450, 2014
[30]. M. Lindorfer, M. Neugschwandtner, C. Platzer, "MARVIN: Efficient and Comprehensive Mobile App Classification Through Static and Dynamic Analysis" 2015 IEEE 39th Annual Computer Software and Applications Conference (COMPSAC) pp. 422-433, 2015
[31]. N. Peiravian, X. Zhu, "Machine Learning for Android Malware Detection Using Permission and API Calls" 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, pp. 300 – 305, 2013
[32]. R. Pirscoveanu, S. Hansen, T. Larsen, et al. "Analysis of malware behavior: Type classification using machine learning" 2015 IEEE International Conference on Cyber Situational Awareness, Data Analytics and Assessment (CyberSA), pp.1-7, 2015
[33]. M .Amin, M. Zaman, M. Hossain, et al. "Behavioral malware detection approaches for Android", 2016 IEEE International Conference on Communications (ICC), pp. 1-6, 2016
[34]. L. Apvrille, A. Apvrille. "Identifying Unknown Android Malware with Feature Extractions and Classification Techniques" TrustCom, pp. 182-189, 2015
[35]. X. Liu, J. Liu, W. Wang, "Exploring sensor usage behaviors of Android applications based on data flow analysis". IPCCC 2015: 1-8.
[36]. D. Su, W. Wang, X. Wang, J. Liu, "Anomadroid: profiling Android applications' behaviors for identifying unknown malapps", 15th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (IEEE TrustCom), 2016
[37]. X. Liu, S. Zhu, W. Wang, J. Liu, "Alde: Privacy Risk Analysis of Analytics Libraries in the Android Ecosystem". 12th EAI International Conference on Security and Privacy in Communication Networks (SecureComm), 2016
[38]. V. Vapnik, "The nature of static learing theory" Springer, 2000
[39]. J. Quinlan, "Introduction of decision tree," Machine learning, vol. 1, no. 1, pp. 81-106, 1986
[40]. L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, pp. 5-32, 2001
[41]. T. Bayes, "An essay towards solving a problem in the doctrine of chances," Philosophical Transactions of the Royal Society, vol. 53, pp. 370-418, 1763
[42]. E. Fix and J. L, Hodges, "Discriminatory analysis: Nonparametric discrimination: Small sample performance," Technical Report Project 21-49-004, Report Number 11, 1952
[43]. Anzhi, http://www.anzhi.com/
[44]. VirusTotal , https://www.virustotal.com/
[45]. Pedregosa et al, "Scikit-learn: Machine Learning in Python," JMLR 12, pp. 2825-2830, 2011.

Authors' Short Biography

Wei Wang is currently associate professor in the School of Computer and Information Technology, Beijing Jiaotong University, China. He earned his Ph.D. degree in control science and engineering from Xi'an Jiaotong University, China, in 2006. He was a postdoctoral researcher in University of Trento, Italy, from 2005-2006. He was a postdoctoral researcher in TELECOM Bretagne and in INRIA, France, from 2007-2008. He was a European ERCIM Fellow in Norwegian University of Science and Technology (NTNU), Norway, and in Interdisciplinary Centre for Security, Reliability and Trust (SnT), University of Luxembourg, from 2009-2011. He visited INRIA, ETH, NTNU, CNR, and New York University Polytechnic. He is young AE of Frontiers of Computer Science Journal. He has authored or co-authored over 50 peer-reviewed papers in various journals and international conferences. His main research interests include mobile, computer and network security.

Yuanyuan Li is currently a M.S. student in the School of Computer and Information Technology, Beijing Jiaotong University, China. She received her B.S. degree from Beijing Jiaotong University, China, in 2015. Her main research interests lie in mobile security.

Xing Wang is currently a Ph.D. student in the School of Computer and Information Technology, Beijing Jiaotong University, China. He received his B.S. degree from Beijing Jiaotong University, China, in 2009. He visited King Abudullah University of Science and Technology (KAUST) from January- April 2014. His main research interests lie in mobile security.

Jiqiang Liu received his B.S. (1994) and Ph.D. (1999) degree from Beijing Normal University. He is currently a Professor at the School of Computer and Information Technology, Beijing Jiaotong University. He has published over 70 scientific papers in various journals and international conferences. His main research interests are trusted computing, cryptographic protocols, privacy preserving and network security.

Xiangliang Zhang is currently an assistant professor and directs the Machine Intelligence and kNowledge Engineering (MINE) Laboratory (http://mine.kaust.edu.sa) in the Division of Computer, Electrical and Mathematical Sciences & Engineering, King Abdullah University of Science and Technology (KAUST). She was a European ERCIM research fellow in the Department of Computer and Information Science, NTNU, Norway, from April–August 2010. She earned her Ph.D. degree in computer science from INRIA-University Paris-Sud 11, France, in July 2010. She has authored or co-authored over 70 refereed papers in various journals and conferences. Her main research interests and experiences are in diverse areas of machine intelligence and knowledge engineering.

***Biographies (Photograph)**

***Biographies (Photograph)**

Click here to download high resolution image

***Biographies (Photograph)**

***Biographies (Photograph)**

***Biographies (Photograph)**

Click here to download high resolution image