Efficient Android Malware Detection Using API Rank and Machine Learning

Jaemin Jung[1], Hyunjin Kim[1], Seong-je Cho[1], Sangchul Han[2*] and Kyoungwon Suh[3]
[1]Dankook University, Yongin, Republic of Korea
{snorlax, khj0417, sjcho}@dankook.ac.kr
[2]Konkuk University, Chungju, Republic of Korea
schan@kku.ac.kr
[3]Illinois State University, Normal IL, United States of America
kwsuh@ilstu.edu

Abstract

As more and more sophisticated Android malwares appear in the online markets, accurate malware detection becomes an important issue in the Android ecosystem. This paper proposes a machine learning based Android malware detection technique that uses ranked Android APIs as machine learning features. First, our technique extracts the information of API invocation from APK files, then produces two ranked lists of APIs frequently used by benign apps and malwares respectively. After filtering out the APIs common to the both lists, we merge the two lists into a single list. We apply three classifiers, random forests (RF), k-nearest neighbor (k-NN), and logistic regression (LR) on a dataset of 60,243 apps using the merged list as the features of the classifiers. Our evaluation results show that the RF classifier can achieve the highest accuracy of 97.47 $\sim$ 98.87% with very low false positive rate (0.99 $\sim$ 2.38%) among them.

keywords: API call, Benign APIs, Malicious APIs, Android malware, Machine Learning, Ranked API list

1 Introduction

Recently many security researchers and engineers attempt to apply machine learning techniques to detect Android malware [3, 7, 9, 12, 13, 14]. Many of them use Android application programming interfaces (APIs) invoked by apps, permissions, and intents of apps as the features for machine learning [3, 7, 9, 12]. These features need to be selected so that they are a representative set of features from which a learning algorithm can construct a classification model for Android malware detection. This process, called feature selection, is an important problem for the performance of machine learning [3, 13, 14].

This paper proposes a machine learning-based Android malware detection technique. In this technique, we use the information of API invocation as the feature. In our study the operating system is Android 7.0 Nougat (API level 24). Android 7.0 supports approximately 133,000 APIs. We obtain the API list from *android.jar* in the Android SDK by using Java reflection. *android.jar* contains Android APIs and is referenced by all Android applications when they are built. By analyzing 60,243 apps, we find out that about 57.24% of the 133,000 APIs are rarely used. This implies that 56,870 Android APIs are commonly used by Android apps. Unfortunately, so many features often lead to huge overheads in both training and testing stages. In addition, if there is too much irrelevant and redundant data, learning during the training stage is more difficult.

Journal of Internet Services and Information Security (JISIS), volume: 9, number: 1 (February 2019), pp. 48-59
*Corresponding author: Department of Software Technology, Konkuk University, 268, Chungwon-daero, Chungju-si, Chungcheongbuk-do, Republic of Korea, Tel: +82-43-840-3605

Our technique identifies a much less but salient subset of Android APIs for learning. By analyzing many benign apps and malwares, we produce two ranked lists of APIs that are frequently used by benign apps and malwares respectively. After filtering out the APIs common to the both lists, we merge the two lists into a single list *Top_N_combined_API_list* where N is a parameter. This list is a smaller but equally effective subsets of the whole API list, which can be used as the features in machine learning algorithm to classify Android apps as benign or malicious one. We adopt three classifiers; random forests (RF) [4, 18], k-nearest neighbor (k-NN) [6, 11], and logistic regression (LR) [8, 15] on a dataset of 60,243 apps using the *top_N_combined_API_lists* as the features of the classifier. The evaluation results show that among the three classifiers the RF classifier can achieve the highest accuracy of $97.5 \sim 98.9\%$ with very low false positive rate ($0.86 \sim 2.41\%$).

This paper extends our previous work [10] where benign ranked API list and malicious ranked API list are not merged, only top 50 APIs are used as the features, and only RF is applied. The work is extended as follows. First, we merge the two ranked API lists and APIs commonly used in both benign and malicious apps are removed. Second, we measure the performance of the proposed technique while varying the values of the parameter *N*. Third, we add two new classifiers, k-NN and LR.

The rest of the paper is organized as follows. Section 2 covers our machine learning based malware detection technique. Section 3 explores the performance of the malware detection technique. Section 4 covers the related work. Section 5 concludes the paper and presents future directions.

2 Proposed Technique

2.1 Background

The Android platform provides the framework API that apps can use to interact with the underlying Android system. The framework API consists of a core set of packages and classes. Because most apps use many APIs, API calls of each app are suitable features for machine learning algorithms to characterize and differentiate malicious apps from benign ones [12]. We reverse engineer APK file and extract API calls of each app using the existing *DexDump* tool that is a part of Android Development Tool (ADT) bundle [1]. DexDump converts the dex file inside the APK into assembly code. By analyzing the assembly code, we can determine what APIs are used.

In a similar way used in [12], we can represent each Android app as binary vector of APIs, namely, a vector A for an app where $A_i = 1$ if and only if the ith API is used in the app and $A_i = 0$ if the app does not use the ith API. We generate such vector for each app in dataset, then construct ranked API lists, i.e., *benign_API_list* and *malicious_API_list*, based on the vectors.

We use RF, k-NN, and LR as the machine learning algorithms for classifying sample apps as benign or malicious. RF has several advantages [4, 18]: (1) it does not require special preprocessing of input; (2) it can deal with large numbers of training instances, missing values, and irrelevant features; (3) because it uses the average of multiple trees, it is less sensitive to outliers in the training set and does not suffer from overfitting compared to using a single decision tree; (4) both training and prediction phases are both fast.

k-NN [6, 11] is a nonparametric classification method. It does not assume any parametric form for the distribution of measured random variables. Due to the flexibility of the nonparametric model, it is usually a good classifier for many situations where the joint distribution is unknown, or hard to model parametrically. Another merit of k-NN is that missing values can be easily imputed. In addition, k-NN has the advantage over artificial neural network (ANN) in terms of the ability to add more data to the model without retraining, and infer the relative importance of the selected features based on their respective weights.

LR [8, 15] is a widely used statistical modeling where the probability of an outcome is related to a series of potential predictor variables. LR model can be used to model complex nonlinear relationships between independent and dependent variables. The model complexity is low in LR, especially when no or few interaction terms and variable transformations are used. Overfitting is less of an issue in this case. Performing variable selection is a way to reduce a model's complexity and consequently decrease the risk of overfitting. However, this may cause a loss in the model's flexibility.

2.2 Malware Detection Using Top Combined API list

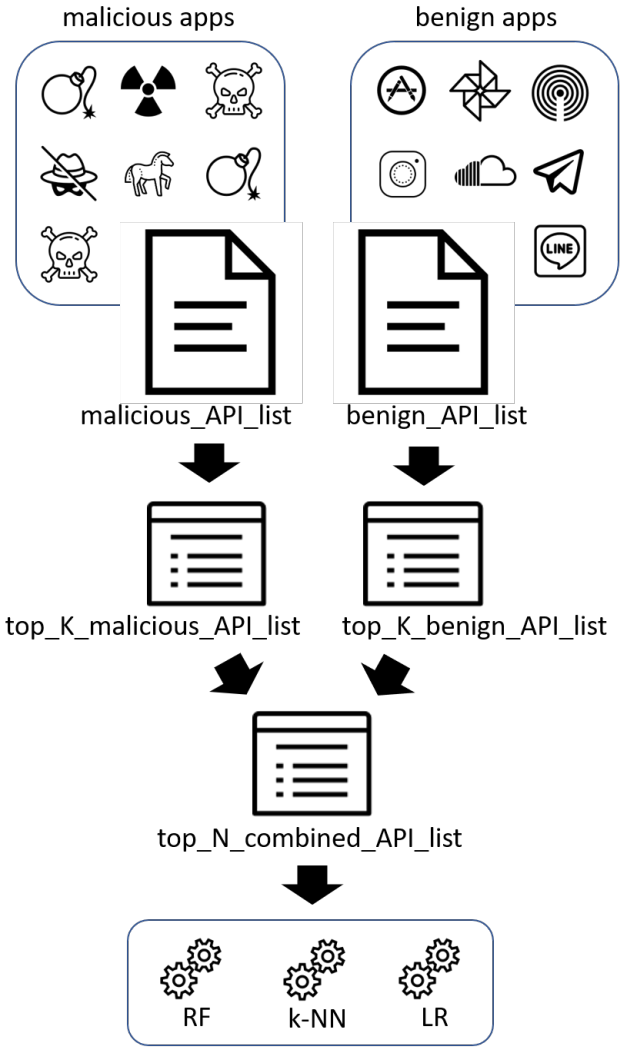

Figure 1: Overview of our proposed technique

Figure 1 shows the overview of our proposed technique. We collect both benign apps and malicious apps that are available in the online Android markets and well-known Android malware dataset. The first dataset, namely benign dataset, is comprised of all benign apps and the second dataset, namely malicious dataset, is comprised of all malicious apps. By analyzing the Android APIs called by each individual app in the two datasets, we first obtain two ranked lists of Android APIs that are commonly used in benign apps (what we call *benign_API_list*) and in malicious apps (what we call *malicious_API_list*).

More specifically, the rank of each API in *benign_API_list* is determined by the total count of benign apps in our training dataset that uses the given API. For example, if an API is used by only one app in our dataset, the API will be ranked the lowest in the ranked list. On the other hand, if an API is used by all apps at least once, the API will be ranked the highest in the ranked list. The rank of each API in *malicious_API_list* is also determined in a similar way. As we will show in the following, the main reason that we create ranked list instead of unranked list is to select a subset of top-K APIs (i.e., most commonly used APIs), in which K is the tunable parameter that we can choose.

From *benign_API_list* and *malicious_API_list*, we create *top_K_benign_API_list* and *top_K_malicious_API_list* by simply choosing the first *K* APIs in the ranked lists, respectively. On the two lists, we conduct the following experimental studies:

(i) We check whether if there is any correlation between *top_K_benign_API_list* and *top_K_malicious_API_list*.

(ii) We construct *top_N_combined_API_list* by combining *top_K_benign_API_list* and *top_K_malicious_API_list*, and filtering out any common APIs overlapped in the two top K API lists, where *N* can be ranged from *K* to *2K*.

(iii) Using the APIs in *top_N_combined_API_list* as the features, we run the three classifiers, RF, k-NN, and LR to detect malwares and measure their accuracy.

(iv) We analyze which classifier is the most efficient for classifying Android apps as benign or malicious using the popular API calls.

The reasons that we employ a static analysis-based machine learning technique are as follows:

1) Static analysis can achieve high code coverage and requires little resources compared to dynamic analysis [4];

2) Machine learning techniques can detect new/unknown malware and do not require a database of malware signatures compared to traditional signature-based malware detection approaches [12];

3) A classifier should be robust and lightweight for classifying Android apps as malicious or benign [3].

We select *top_N_combined_API_list* as the best features for machine learning that can distinguish malware from benign apps and detect new/unknown malicious apps effectively and efficiently. We expect that the *combined_API_list* include the representative APIs for both malicious and benign apps.

3 Experiment Result

In this section, we present the process of feature selection, i.e., the construction of *top_N_combined_API_list*, and the classification results of the three classifiers while varying *K*=10, 20, . . . , 100.

3.1 Datasets

We collected more than 31,000 Android apps from various Android markets including Google Play (https://play.google.com/store), Amazon AppStore(https://www.amazon.com/amazonappstoreapp), and APKPure (https://apkpure.com/) from December in 2016 to February in 2017. By relying on antivirus tools provided by VirusTotal website [2], we first classified each of the collected Android apps as a

benign app or a malicious app. 30,159 of the 31,000 apps were identified as benign apps by VirusTotal. The 30,159 benign apps were selected as the dataset of benign apps in our experiment. In addition, we collected 30,084 malicious Android apps from AMD [16] and Drebin [5] datasets which are well-known datasets and have been used in several research papers. The numbers of malicious apps we collected from AMD and Drebin datasets are 24,547 and 5,537, respectively. The collected malicious apps were selected as the dataset of malicious apps in our experiment. From the dataset of 30,159 benign apps and 30,084 malicious apps, we randomly selected 80% of the datasets respectively and used them as training datasets. The remaining 20% of the datasets is used as test datasets.

3.2 Popular APIs

By analyzing the Android APIs invoked by each app in the two training datasets, we construct the *benign_API_list* and *malicious_API_list*. Then, from the two ranked lists, we create *top_K_benign_API_list* and *top_K_malicious_API_list* by simply choosing the first K APIs in the ranked lists, respectively, where K is 10, 20, ..., 100. For the interest of space, we present only the first 10 APIs of the two top K lists in Table 1. Interestingly, we observe that $50 \sim 60\%$ of the two top K ($K \leqq 100$) API lists are common. As shown in Table 1, the top 5 APIs are common in both top 10 API lists. Since these common APIs are irrelevant with respect to the classification, we eliminate these irrelevant feature in the next step.

The *top_N_combined_API_list* is generated by merging *top_K_benign_API_list* and *top_K_malicious_API_list*, and filtering out the common APIs overlapped in the two top K lists. Table 2 shows Top 10 Combined APIs derived from API lists of Table 1. As it shows, the two lists shown in Table 1 are combined after removing 5 common APIs of the both lists. Table 3 shows the number of common APIs overlapped in both top K lists and the values of N in *top_N_combined_API_list*. Note that N means the number of unique APIs in the lists.

Table 1: Top 10 Benign and Malicous APIs

	Top 10 benign APIs	Top 10 malicious APIs
1	Ljava/lang/Object;.<init>:()V	Ljava/lang/Object;.<init>:()V
2	Ljava/lang/StringBuilder;.toString:()Ljava/lang /String;	Landroid/app/Activity;.<init>:()V
3	Ljava/lang/StringBuilder;.append:(Ljava/lang /String;)Ljava/lang/StringBuilder;	Ljava/lang/String;.equals:(Ljava/lang/Object;)Z
4	Landroid/app/Activity;.<init>:()V	Ljava/lang/StringBuilder;.toString:()Ljava/lang /String;
5	Ljava/lang/StringBuilder;.<init>:()V	Ljava/lang/StringBuilder;.append:(Ljava/lang /String;)Ljava/lang/StringBuilder;
6	Ljava/lang/String;.equals:(Ljava/lang/Object;)Z	Landroid/content/BroadcastReceiver;.<init>:()V
7	Ljava/lang/StringBuilder;.append:(I)Ljava/lang /StringBuilder;	Landroid/content/Context;.getSystemService: (Ljava/lang/String;)Ljava/lang/Object;
8	Ljava/util/ArrayList;.<init>:()V	Landroid/content/Intent;.<init>:(Landroid /content/Context;Ljava/lang/Class;)V
9	Ljava/util/Iterator;.hasNext:()Z	Landroid/net/Uri;.parse:(Ljava/lang/String;) Landroid/net/Uri;
10	Ljava/util/Iterator;.next:()Ljava/lang/Object;	Ljava/lang/System;.currentTimeMillis:()J

Table 2: Top 10 Combined APIs

Label	Top K combined API
Benign	Ljava/lang/StringBuilder;.<init>:()V
Benign	Ljava/lang/StringBuilder;.append:(I)Ljava/lang/StringBuilder;
Benign	Ljava/util/ArrayList;.<init>:()V
Benign	Ljava/util/Iterator;.hasNext:()Z
Benign	Ljava/util/Iterator;.next:()Ljava/lang/Object;
Malicious	Landroid/content/BroadcastReceiver;.<init>:()V
Malicious	Landroid/content/Context;.getSystemService:(Ljava/lang/String;)Ljava/lang/Object;
Malicious	Landroid/content/Intent;.<init>:(Landroid/content/Context;Ljava/lang/Class;)V
Malicious	Landroid/net/Uri;.parse:(Ljava/lang/String;)Landroid/net/Uri;
Malicious	Ljava/lang/System;.currentTimeMillis:()J

Table 3: Values of N in *Top_N_combined_API_list*

K	No. of Common APIs	N(=2(K - No. of Common APIs))
10	5	10
20	12	16
30	15	30
40	21	38
50	25	50
60	35	50
70	41	58
80	47	66
90	51	78
100	56	88

3.3 Evaluation Metrics

In this paper, we identify Android malicious app as the true instance and benign app as the false instance. To evaluate the effectiveness of proposed method, we measure *accuracy*, *false positive (FP)*, *false negative (FN)*, *true positive (TP)*, and *true negative (TN)* in our experiments. TP is the number of malwares that are correctly detected, FP is the number of benign apps that are incorrectly detected as malwares, FN is the number of malwares that are not detected (predicted as benign apps), and TN is the number of benign apps that are correctly classified. In our test dataset, TP+FN = 6017, and FP+TN=6032. The *accuracy*, *false positive rate*, and *false negative rate* are as follows:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

False positive rate = FP / (TN + FP)

False negative rate = FN / (TP + FN)

Table 4: Results with *Top_N_Combined_API_list* as Features in RF

K	Measure Metrics					
	TP	FP	FN	TN	Acc.	Avg. of Acc.
10	5860	143	172	5874	97.39%	97.47%
20	5891	130	141	5887	97.75%	97.83%
30	5906	94	126	5923	98.17%	98.20%
40	5914	79	118	5938	98.37%	98.46%
50	5929	77	103	5940	98.51%	98.52%
60	5907	70	125	5947	98.38%	98.50%
70	5929	68	103	5949	98.58%	98.58%
80	5932	71	100	5946	98.58%	98.66%
90	5942	61	90	5956	98.75%	98.78%
100	5953	60	79	5957	98.85%	98.87%

Table 5: Results with *Top_N_Combined_API_list* as Features in LR

K	Measure Metrics					
	TP	FP	FN	TN	Acc.	Avg. of Acc.
10	4618	610	1414	5407	83.20%	84.11%
20	5122	550	910	5467	87.88%	88.12%
30	5277	416	755	5601	90.28%	90.44%
40	5299	289	733	5728	91.52%	91.81%
50	5419	276	613	5741	92.62%	92.77%
60	5497	268	535	5749	93.34%	93.30%
70	5546	254	486	5763	93.86%	94.03%
80	5568	211	464	5806	94.40%	94.47%
90	5645	172	387	5845	95.36%	95.46%
100	5646	172	386	5845	95.37%	95.54%

Table 6: Results with *Top_N_Combined_API_list* as Features in k-NN

K	Measure Metrics					
	TP	FP	FN	TN	Acc.	Avg. of Acc.
10	5644	202	388	5815	95.10%	95.24%
20	5691	164	341	5853	95.81%	95.83%
30	5733	119	299	5898	96.53%	96.78%
40	5785	121	247	5896	96.95%	97.20%
50	5815	109	217	5908	97.29%	97.49%
60	5770	107	262	5910	96.94%	97.24%
70	5798	104	234	5913	97.19%	97.38%
80	5788	100	244	5917	97.15%	97.30%
90	5819	101	213	5916	97.39%	97.46%
100	5826	86	206	5931	97.62%	97.62%

3.4 Evaluation Results

We repeated the following experiment 5 times for each K, in which K=10, 20, . . . , 100:

1) Randomly select training data (80%) and test data (20%)
2) Construct *top_N_combined_API_list*
3) Find optimal hyperparameters using grid search
4) Train and test for each classifier

Table 4, 5, and 6 show the results of the three classifiers. The column *TP*, *FP*, *FN*, *TN*, and *Acc* show the measurement of the first experiment and the column *Avg. of Acc* shows the average accuracy of the five experiments. As presented in Table 4 that summarizes the performance of RF, *false negative rate (FNR)* was 1.31 $\sim$ 2.85%, and *false positive rate (FPR)* was 0.99 $\sim$ 2.38%. The average accuracy has ranged from 97.47% to 98.87%. As to LR (Table 5, the average accuracy was 84.11 $\sim$ 95.54%, *FNR* was 6.40 $\sim$ 23.44%, and *FPR* was 2.86 $\sim$ 10.14%. Table 6 shows the performance of k-NN (k=9), where the average accuracy has ranged from 95.24% to 97.62%, *FNR* is 3.42 $\sim$ 6.43%, and *FPR* is 1.43 $\sim$ 3.36%. Overall, RF outperforms other classifiers.

Figure 2 shows the average accuracy of the classifiers for each K. On the whole, the classifiers perform better as K increases, that is, as more APIs are used as features. In particular, RF still works effectively with high accuracy when a relatively small value is used for K. This implies that RF can effectively detect Android malware with a small number of APIs if the APIs are selected properly.

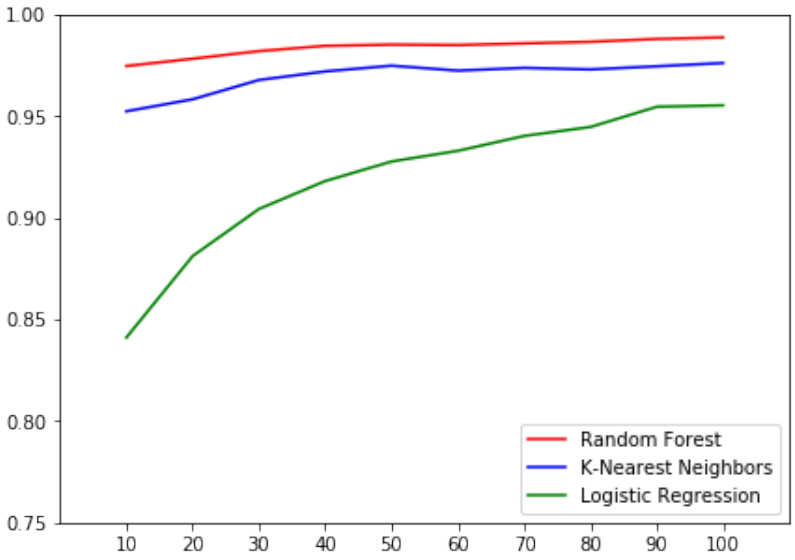

Figure 2: Average accuracy of three experiments.

4 Related Works

Peiravian and Zhu combined permissions and API calls and used machine learning methods to classify sample Android apps as benign or malicious apps [12]. By using permissions and API calls as features to characterize each app, they could learn a classifier to identify whether an app is benign or malicious. Their approach was validated by carrying out experiments on real-world apps with 1,260 malicious samples and 1,250 benign samples. In the experiments, they examined three classification methods: Support

Vector Machines (SVM), Decision tree (DT) of J48, and Bagging, and used 10-fold cross validation. J48 is a WEKA implementation of the C4.5 algorithm for the decision tree. Their approach achieved good accuracy rate up to 96.88% with 1,456 features (130 permissions + 1,326 APIs).

Aafer [3] extracted relevant features to malware behavior captured at API level. More specifically, they analyzed API level information to select the best features that distinguish between malicious and benign. They generated the set of APIs used within each app, and performed a frequency analysis to list out the ones which were more frequent in the malware than in the benign apps. They focused on critical API calls (or dangerous APIs), package level information, and the parameters of the API calls. Their analysis showed that 71% of the benign apps contained at least one advertisement package, and the packages often exhibited some suspicious behavior. Meanwhile, certain frequent APIs in the malicious samples did not differ from those of the benign sample. To increase the difference of these specific APIs between the malicious and benign samples, data flow analysis on the APIs was performed in order to recover the parameters that had been passed to them. Their study aimed to identify at what package level a certain API is invoked. They evaluated different classifiers using the generated feature set. They could achieve an accuracy as high as 99% and a FPR as low as 2.2% using the k-NN classifier when the top 20 used parameters was added the top 169 APIs.

Goyal [9] devised SafeDroid, an open source distributed service to detect Android malware by combining static analysis and machine learning technique. SafeDroid inspects the device for installed apps and classifies apps as benign or malicious. The backend system was implemented as a micro-service which provides classification, based on the API calls of the examined app. SafeDroid identified a list of 743 APIs that described malicious behavior. Using Correlation Attribute Evaluation (CAE) method for feature selection, the 300 top ranked features were selected from the 743 features. SafeDroid identified the top 108 packages used in malicious apps more than in benign apps. The performance of SafeDroid was evaluated using three different classifiers over a dataset of real application. The results were averaged over 10-folds of cross validation. Goyal [9] found out that the Random Forest performed the best with an accuracy of 99.51% and a FPR of 0.017.

Alam and Vuong [4] applied RF classifier on a dataset of 48,919 points of 42 dynamic features. The dynamic features were obtained from emulating user action using adb-monkey on unrooted Android device emulators. The predominant features included the Binder APIs, memory and CPU measurements, battery usage, etc. Their objective was to measure the accuracy of RF in classifying Android app behavior to classify apps as malicious or benign. The experimental results based on 5-fold cross validation of their dataset showed that RF achieved an accuracy of over 99% in general, with less for forests of 20 trees or more. Since they used the dataset collected from an Android emulator, the work should be extended to gather dataset based on actual device usage.

Wu et al. [17] proposed a static feature-based mechanism for detecting the Android malware, where the static feature information includes permissions, deployment of components, Intent messages passing and API calls for characterizing the Android apps behavior. They tried to apply different kinds of clustering algorithms to recognize different intentions of Android malware as well as enhance the malware modeling capability. They have also developed a system, called DroidMat, which first extracts the information (e.g., permissions and Intent messages passing, etc) from each app's manifest file, and regards Android components as entry points drilling down for tracing API Calls related to permissions. Then, DroidMat applies K-means algorithm that enhances the malware modeling capability. Finally, it uses k-NN algorithm to classify the apps as benign or malicious. The experiment results showed that DroidMat could achieved up to 97.87 percentage points in accuracy.

Table 7 shows the differences between our approach and the previous studies. Compared to the previous studies, our approach has several advantages: use of real dataset, single type of features, no need of dynamic analysis, a much smaller feature set, and utilization of popular benign APIs.

Table 7: Comparison with Existing Works

	Dataset	Features	Classifier	Accuracy
Peiravian & Zhu [12]	1,260 malicious apps, 1,250 benign apps	1,326 APIs, 130 Permissions	SVM, J48, Bagging	93.54∼96.88%, 92.36∼94.46%, 93.60∼96.39%
Aafer et al. [3]	3,987 malware apps, 500 benign apps	169 critical APIs, 20 API Parameters, Package info	k-NN, SVM, ID3 DT, C4.5 DT	99.1%, 96.5%, 97.9%, 95.5%
Goyal et al. [9]	4,554 malware apps, 20,446 benign apps	300 top ranked features of 743 malicious APIs	RF (trees=10), Liner SVM, k-NN(k=20)	96.95∼99.51%, 96.04∼97.27%, 96.46∼98.74%
Alam & Vuong [4]	1,330 malicious apps, 407 benign apps	42 dynamic features	RF (trees= 10, 20, 40, 80, 160)	Over 99%
Wu et al. [17]	238 malicious apps, 1,500 benign apps	Intents, Permissions, API calls	EM + k-NN, K-means + k-NN, EM + NB, K-means + NB	Up to 97.87%
Our approach	30,084 malware apps, 30,159 benign apps	N popular Combined APIs (K=10∼100)	RF (trees =100), LR, k-NN (k=9)	97.46∼98.87%, 84.11∼95.54%, 95.24∼97.62%

5 Conclusion and Future Work

In this paper, we proposed a machine learning-based technique to detect Android malwares efficiently by analyzing the two ranked lists of APIs used by apps that we collected from online Android markets and known malware data sets, respectively. The rank of each API in the lists was determined by the total number of apps that call the corresponding API. Specifically, we selected the top K APIs in the ranked list obtained from benign apps and the top K APIs in the ranked list obtained from malicious apps. Then, we created *top_N_combined_API_list* by combining *top_K_benign_API_list* and *top_K_malicious_API_list*, and filtering out some common APIs overlapped in the both top K API lists, where N ($K \leqq N \leqq 2K$) is 10, 16, 30, 38, 50, 50, 58, 66, 78, 88. We used RF, LR, and k-NN as the underlying machine learning classifiers for Android malware detection and the *top_N_combined_API_lists* were used as features for the classifiers. In all cases, accuracy, FP, and FN were measured. The RF classifier could achieve very high detection accuracy (i.e., 97.47 $\sim$ 98.87%) with very small numbers of distinct APIs. In the future, we plan to classify malicious Android apps into different malware families using the RM, LR, and k-NN classifiers. In addition, we plan to develop malware detection methodologies based on SVM and artificial neural network (ANN) and also compare the effectiveness of different machine learning approaches for the purpose of malware detection and family classification.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science and ICT (no. 2018R1A2B2004830) and the MSIT(Ministry of Science and ICT), Korea, under the National Program for Excellence in SW supervised by the IITP(Institute for Information & communications Technology Promotion)(2017-0-00091).

References

[1] Android developers. `https://developer.android.com/`, [Online; accessed on February 25, 2019].

[2] Virustotal. `https://www.virustotal.com/`, [Online; accessed on February 25, 2019].

[3] Y. Aafer, W. Du, and X. Zhu. Droidapiminer: Mining api-level features for robust malware detection in android. In *Proc. of the 9th International Conference on Security and Privacy in Communication Systems (SecureComm'13), New South Wales, Australia*, pages 86–103. Springer, September 2013.

[4] M. Alam and S. Vuong. Random forest classification for detecting android malware. In *Proc. of the 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing (GreenCom-iThings-CPSCom'13), Beijing, China*, pages 663–669. IEEE, August 2013.

[5] D. Arp, M. Spreitzenbarth, M. Huebner, H. Gascon, and K. Rieck. Drebin: Effective and explainable detection of android malware in your pocket. In *Proc. of the 2014 Network and Distributed System Security Symposium (NDSS'14), San Diego, California, USA*, pages 23–26. Internet Society, February 2014.

[6] W. Cedeño and D. Agrafiotis. Using particle swarms for the development of qsar models based on k-nearest neighbor and kernel regression. *Journal of Computer-Aided Molecular Design*, 17(2-4):255–263, February 2003.

[7] P. Chan and W. Song. Static detection of android malware by using permissions and api calls. In *Proc. of the 2014 International Conference on Machine Learning and Cybernetics (ICMLC'14), Lanzhou, China*, pages 82–87. IEEE, July 2014.

[8] S. Dreiseitl and L. Ohno-Machado. Logistic regression and artificial neural network classification models: a methodology review. *Journal of Biomedical Informatics*, 35(5-6):352–359, October 2002.

[9] R. Goyal, A. Spognardi, N. Dragoni, and M. Argyriou. Safedroid: a distributed malware detection service for android. In *Proc. of the 9th International Conference on Service-Oriented Computing and Applications (SOCA'16), Macau, China*, pages 59–66. IEEE, November 2016.

[10] J. Jung, H. Kim, D. Shin, M. Lee, H. Lee, S. Cho, and K. Suh. Android malware detection based on useful api calls and machine learning. In *Proc. of the 2018 IEEE 1st International Conference on Artificial Intelligence and Knowledge Engineering (AIKE'18), Laguna Hills, California, USA*, pages 175–178. IEEE, September 2018.

[11] S. Li, E. Harner, and D. Adjeroh. Random knn feature selection-a fast and stable alternative to random forests. *BMC Bioinformatics*, 12(1):450, November 2011.

[12] N. Peiravian and X. Zhu. Machine learning for android malware detection using permission and api calls. In *Proc. of the 25th International Conference on Tools with Artificial Intelligence (ICTAI'13), Herndon, Virginia, USA*, pages 300–305. IEEE, November 2013.

[13] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, J. Nieves, P. Bringas, and G. Álvarez. Mama: Manifest analysis for malware detection in android. *Cybernetics and Systems*, 44(6-7):469–488, August 2013.

[14] J. Saxe and K. Berlin. Deep neural network based malware detection using two dimensional binary program features. In *Proc. of the 10th International Conference on Malicious and Unwanted Software (MALWARE'15), Fajardo, Puerto Rico*, pages 11–20. IEEE, October 2015.

[15] J. Tu. Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes. *Journal of Clinical Epidemiology*, 49(11):1225–1231, November 1996.

[16] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou. Deep ground truth analysis of current android malware. In *Proc. of the 14th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'17), Bonn, Germany*, pages 252–276. Springer, July 2017.

[17] D. Wu, C. Mao, T. Wei, H. Lee, and K. Wu. Droidmat: Android malware detection through manifest and api calls tracing. In *Proc. of the 7th Asia Joint Conference on Information Security (AsiaJCIS'12), Tokyo, Japan*, pages 62–69. IEEE, August 2012.

[18] S. Yerima, S. Sezer, and I. Muttik. High accuracy android malware detection using ensemble learning. *IET Information Security*, 9(6):313–320, October 2015.

Author Biography



Jaemin Jung received the B.S. degree in Dept. of Software Science from Dankook University, Korea, in 2018. He is currently a M.E. student in Computer Science and Engineering at Dankook University, Korea. His research interests include mobile security and machine learning.



Hyunjin Kim received the B.S. degree in Dept. of mathematics and computer science from Dankook University, Korea, in 2017. She is currently a M.S. student in Data Science at Dankook University, Korea. Her research interests include machine learning and recommendation system.



Seong-je Cho received the B.E., M.E. and Ph.D. degrees in Computer Engineering from Seoul National University in 1989, 1991 and 1996, respectively. In 1997, he joined the faculty of Dankook University, Korea, where he is currently a Professor in Department of Computer Science and Engineering (Graduate school) and Department of Software Science (Undergraduate school). He was a visiting research professor at Department of EECS, University of California, Irvine, USA in 2001, and at Department of Electrical and Computer Engineering, University of Cincinnati, USA in 2009 respectively. His current research interests include computer security, mobile app security, operating systems, and software intellectual property protection.



Sangchul Han received his B.S. degree in Computer Science from Yonsei University in 1998. He received his M.E. and Ph.D. degrees in Computer Engineering from Seoul National University in 2000 and 2007, respectively. He is now a professor of Dept. of Software Technology at Konkuk University. His research interests include real-time scheduling, and computer security.



Kyoungwon Suh received the B.S. and M.S. degrees in computer engineering from Seoul National University, Seoul, Korea, in 1991 and 1993, respectively, the M.S. degree in computer science from Rutgers University, New Brunswick, NJ, in 2000, and the Ph.D. degree in computer science from the University of Massachusetts, Amherst, in 2007. He is currently a Professor of Computer Science with Illinois State University, Normal. His research interests include network measurement/analysis/inference, wireless networks, smart handheld devices, content distribution networks, big data analysis, privacy, and security.