AndMFC: Android Malware Family Classification Framework

Sercan Türker
Department of Computer Engineering
University of Hacettepe
Ankara, Turkey
n14323115@cs.hacettepe.edu.tr

Ahmet Burak Can
Department of Computer Engineering
University of Hacettepe
Ankara, Turkey
abc@cs.hacettepe.edu.tr

***Abstract*—As the popularity of Android mobile operating system grows, the number of malicious software have increased extensively. Therefore, many research efforts have been done on Android malware analysis. Besides detection of malicious Android applications, recognizing families of malwares is also an important task in malware analysis. In this paper, we propose a machine learning-based classification framework that classifies Android malware samples into their families. The framework extracts requested permissions and API calls from Android malware samples and uses them as features to train a large set of machine learning classifiers. To validate the performance of our proposed approach, we use three different malware datasets. Our experimental results show that all of the tested models classify malwares efficiently. We also make a study of detecting unknown malwares that never seen before and we notice that our framework detects these malwares with a high accuracy.**

Keywords— Android Malware, Malware Classification, Family Classification, Static Analysis, Machine Learning

I. Introduction

Smart phones have become an important part of our lives in the last decade. Among the mobile operating systems on smart phones, Android has become the most popular one and has dominated the market with a high sales rate. In the second quarter of 2018, 88 percent of all smart phones sold to end users are working on the Android operating system [1]. Being the most popular mobile operating system, Android has been more targeted than any other mobile operating systems by attackers [2]. Therefore, Android malware analysis has become one of the most interesting topics for researchers. Although most of the studies are done to classify Android applications as either malicious or benign, classifying malwares into their families is also an important research topic. Since malwares of the same family are expected to exhibit the similar behavior, family classification can be useful in deciding whether the malware is a sort of already known malware. This decision eases malware analysts' work by enabling to focus solely on new malwares that does not belong to any known family, rather than wasting time on analyzing known malware types.

Android malware analysis can be grouped into two main sections: static analysis and dynamic analysis. Static analysis is based on analyzing of application's manifest and code files. Thus, the package file of the application and a tool for parsing this file are enough for the analysis. On the other hand, in dynamic analysis, application's behavior is monitored during its execution in a real or emulated environment. Each method has its own advantages and disadvantages. Implementing static analysis methods is easier than the dynamic approach but static analysis can be prevented with obfuscation techniques. Although dynamic analysis is more robust for obfuscation, it can be defeated by some evasion techniques.

In this paper, we propose an Android malware classification framework (AndMFC) that can recognize families of Android malwares. The framework performs static analysis on malwares and extracts requested permissions and API calls as features. After selecting the most distinguishing features, a set of machine learning classification algorithms, including ensemble and neural network algorithms, are used to classify malwares. In the experiments, Support Vector Machine, Decision Tree, Logistic Regression, K-Nearest Neighbors, Random Forest, AdaBoost, and Multi-Layer Perceptron classifiers are used for classification. To extensively evaluate the performance of these classifiers, three different datasets are used. The results of evaluations show that the extracted features from static analysis are useful to identify malware families and most of the tested classification methods can accurately classify Android malware into the families. In addition to classification, we also examine characteristics of malwares by leveraging the feature set and report the specific behaviors of malware categories. As the final work, we propose an approach for detecting the malwares that does not belong to any known family and classified them as unknown. Overall, the main contributions of this paper are:

- We present an Android malware family classification framework that performs static analysis and open the code for future studies of other researchers[1].
- The framework uses static analysis to classify malware families. The proposed method provides resilience to renaming attacks and partial resilience to dynamic loading attacks, which are used to evade static analysis.
- The proposed framework can use a large set of machine learning classification methods including ensemble and neural network algorithms. It can be expendable with future classification methods.
- We measure the performance of each tested classifier in terms of accuracy, precision, recall, and F1 score values. We report the classification results on AMD, Drebin and UpDroid datasets. It is also shown that the feature set obtained in AMD dataset training is helpful to classify malwares in other datasets.
- We show how the framework can be used on detecting unknown malwares and report detection performance in terms of accuracy, precision, recall and F1 score.
- Examining the frequency of use, we have addressed some of the important features that are useful to characterize malware categories.

[1] The code for the AndMFC framework can be found at : https://github.com/srcntrkr/AndroidFamilyClassification

The rest of the paper is organized as follows. In Section II, the prior researches that have been made to identify and classify Android malware families are reviewed. Section III explains the proposed model in details. The evaluation results are presented in Section IV and finally, we conclude the work in section V.

II. RELATED WORK

Several machine-learning based approaches have been presented in the literature to detect Android malware automatically with the features extracted from static and/or dynamic analysis. Some works like Drebin [8], DroidAPIMiner [9], AppContext [10] and Anastasia [11] perform static analysis for malware detection, whereas Stream [12], MADAM [13], DroidDolphin [14], Caviglione et al [15] and Nancy et al [16] use dynamic features extracted from dynamic analysis. Marvin [17] and F. Yang et al [18] used both static and dynamic features to detect malware. Deep learning based malware detection is also interested in some works like HADM [19], DroidSec [20] and DroidDeep [21].

In addition to malware detection, there are some studies in malware family classification. DroidLegacy [22] system extracts familial signatures from API calls to classify malware into families. RevealDroid [23] performs static analysis that is resilient against basic obfuscation techniques and classifies malware with C4.5 and 1NN classifiers. Their approach observes information flow and sensitive API flow during static analysis. DroidSieve [24] also considers only static and obfuscation resilient features for classification. DroidScribe [25] uses a dynamic approach for malware family classification and extracts features from runtime behavior such as system calls and network transactions. Masseralli et al [26] also proposes a dynamic approach that relies on resource consumption. Dendroid [27] proposes a text-mining method to automatically classify malware samples into families based on the similarity of their code structures. LEILA [28] analyses Java Bytecode that is produced when the source code is compiled and identifies malicious behavior of each malware family. EC2 [29] proposes an algorithm that effectively classifies a malware sample into both large and small families. The system leverages both static and dynamic features and is designed as an ensemble that combines the best of classification and clustering. UpDroid [32] is another work that performs hybrid analysis and classifies the malwares of their own dataset [33].

III. DESIGN AND IMPLEMENTATION

In this section, we describe design and implementation of our framework. The goal of our system is classifying Android malware applications into malware families. Our proposed framework performs static analysis of malwares and extracts a huge number of features from package contents, where some of the informative features represent characterization of malwares. Using some of the selected features, classification of malwares is performed with a set of classifiers. The components of the AndMFC framework are illustrated in Figure 1. One of the important points of the AndMFC is that any feature extraction / classification method can be plugged into the system to make further analysis.

A. *Feature Extraction*

Our framework performs a static analysis and extracts a comprehensive set of features to characterize a malware accurately. To extract these features, a reverse engineering tool called Apktool v2.3.4 [4] is leveraged. Apktool decompiles the malwares and generates an output folder that contains all Java classes in Smali [5] language, the manifest file of the application, resources, libraries and assets. We focus on parsing manifest and Smali files to extract the following features:

- **Permissions**: Permission system is an important security mechanism of Android platform that is used to organize the application's access to sensitive user data (e.g., SMS, Calendar) and system features (e.g., Camera, Microphone). This characteristic makes the requested permissions one of the most used static features in Android malware analysis [6]. Extracting permission features is crucial since they are helpful to address the malware family. One example is that malwares belong to *Svpeng* family lock the mobile device. To achieve this malicious action, they request *SYSTEM_ALERT_WINDOW* permission [7]. Therefore, using this permission as a feature is helpful to determine whether a malware belongs to the Svpeng family or not. With this observation, 278 different permission features are extracted.
- **API Calls:** Android API contains a set of functions used by a software to interact with Android operating system. The functions used by the malware developers can include a clue about the malware's family. For example, some malware families such as *BankBot* and *Triada* include codes to evade dynamic analysis. To achieve this, they check whether the system that runs the application is a real or an emulated system by using some API methods like *GetDeviceId*. Thus, using this API call as a feature can help us identifying a malware's family. A total of 28433 different API call features are extracted regardless of whether they are suspicious or innocuous. Most malware detection methods extract only suspicious calls but AndMFC considers all types of API calls to classify different types of malware families.

Some malware families in the dataset use many techniques to evade static analysis. Renaming is one of them and it is an obfuscation technique that gives a random name to variables, methods and classes. When this technique is applied to an application, the Smali files extracted by the Apktool contains meaningless names instead of real names. However, API calls cannot be renamed and they are always visible in Smali files [7]. Thus, our system is resilient to renaming attacks.

Another technique to evade static analysis is dynamic loading that means a malware load the malicious code from another application. Our framework looks for another application in *assets* folder of Apktool output and analyzes them to extract features as the features of the related malware. In this way, dynamically loaded codes can be taken into account and classification performance can be increased. Because of the fact that loading from the *assets* folder is not the only way of dynamic loading, we can say that we provide a partial resilience to this technique.

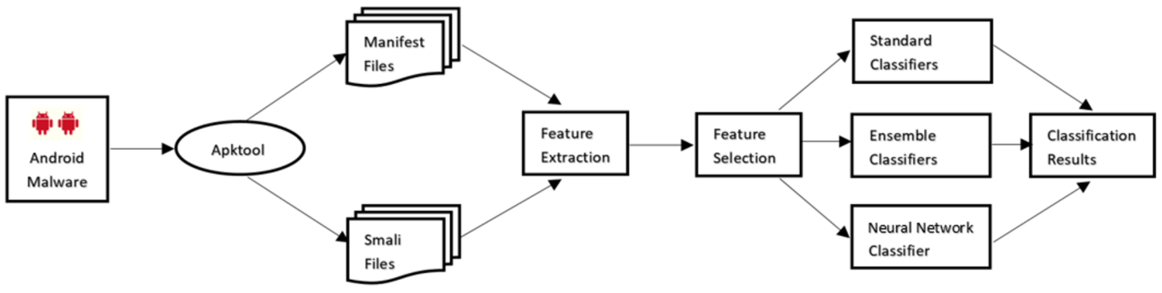

Figure 1: Overview of the AndMFC system

B. *Feature Selection*

As a preprocessing step, we apply a feature ranking method to determine the features that affect the results most. We use AMD [3] dataset and Random Forest classifier to learn the importance of each feature and select the top 1000 features (Due to space limit, the reason why the top 1000 are selected is given in our thesis work [31]). The new feature set contains 42 permission features and 958 API call features. Discarding redundant and irrelevant features reduces model complexity and execution time, while increasing the accuracy of classification.

C. *Classification Models*

We employ several machine learning classifiers to build a promising model for classifying Android malwares into their families. Apart from the standard supervised classifiers such as Support Vector Machine (SVM), Decision Tree (DT), Logistic Regression (LR) and K-Nearest Neighbors (KNN), some ensemble classifiers such as Random Forest (RF) and AdaBoost are used. Since deep learning is a popular approach in machine learning, we also apply Multi-Layer Perceptron (MLP) algorithm in our experiments.

Apart from these classifiers, we combine three standard classifiers (DT, LR and KNN) and create an ensemble Majority Voting (MV) classifier. Each classifier in this ensemble model makes a prediction (vote) and the final prediction is selected as the prediction of the majority of the classifiers.

These classification algorithms have hyper parameters and proper settings of these parameters can affect the classification performance positively. Thus, we perform hyper-parameter optimization on our classifiers through Grid-search and Cross-validation processes implemented in Scikit-learn Machine Learning package [30]. Table I shows the hyper parameters for each classifier.

TABLE I. PARAMETER TUNING VIA GRID-SEARCH AND CROSS-VALIDATION

Classifier	**Best Parameters**
SVM	kernel=’poly’, gamma=0.1, C=1.0
DT	max depth=18, criterion=’entropy’,
LR	penalty=’l2’, C=0.1,
KNN	n_neighbors=5, p=2, metric=’minkowski’
RF	n_estimators=10000
AdaBoost	n_estimators=500, learning_rate=1
MV	Best parameters of DT, LR and KNN
MLP	nEpoch=100, batch_size=100, num_hidden_layers=3, layer_size=[1000,500,250,125,71]

IV. EVALUATION

In this section, we evaluate the overall system and present the classification performance results of each classifier. We use AMD [3] dataset for our experiments and analyze 24467 Android malware samples which belong to 71 different Android malware families and 10 malware categories. We eliminate 86 malware samples because of the execution errors of ApkTool.

After the presentation of the results on AMD dataset, we characterize some malware categories and malware families by examining the frequency of their features. Then, we also validate the performance of our system on Drebin and UpDroid datasets. Finally, we work on classifying *unknown* malware which has not been studied before in our knowledge.

A. *Evaluation Metrics*

To measure the performance of classifiers, we calculate Accuracy (Acc), Macro Precision (MaP), Macro Recall (MaR) and Macro F1 Score (MaF) values. Accuracy is an overall measure of the classifier and corresponds to the proportion of correct results that the classifier achieved. Precision measures how many of the malware samples associated to a family are indeed belongs to that family, while recall measures how many malware samples are correctly classified. Macro Precision and Macro Recall corresponds to the averages of precision and recall values of families. Finally, Macro F1 Score is the harmonic average of the Macro Precision and Macro Recall values.

B. *Classification Results on AMD Dataset*

As the first testing method, we perform 10-fold cross validation on the entire dataset to measure performance of the classification models. As the second testing method, we randomly split the whole dataset into equal sized training and test sets (2-fold). Since the dataset is an unbalanced dataset, we pay attention to get samples from each family with equal ratio and we randomly select half of each family as test data. Classification accuracy results obtained from these two experiments are given in Table II.

As shown in Table II, SVM, AdaBoost and MLP gives better classification performance compared to others. 10-fold cross validation accuracies and 2-fold test accuracies of all classifiers are close to each other. All the tested classifiers are able to achieve more than 96% accuracy. This shows that the extracted features are meaningful to describe malware families of the dataset. SVM gives the best performance among the standard classifiers, while AdaBoost is the best ensemble classifier. MV classifier achieves better performance than individual DT and KNN scores.

TABLE II. CLASSIFICATION ACCURACY SCORES

Classifier	Type	10-fold Accuracy (%)	2-fold Accuracy (%)
SVM	Standard	98.86 (+-0.98)	99.12
DT	Standard	97.12 (+-1.71)	97.27
LR	Standard	98.47 (+-0.96)	98.87
KNN	Standard	96.51 (+-1.44)	96.98
RF	Ensemble	98.18 (+-1.24)	98.42
AdaBoost	Ensemble	98.68 (+-1.16)	99.08
MV	Ensemble	98.31 (+-1.40)	98.63
MLP	Neural Net	98.67 (+-0.96)	99.00

In order to analyze these results further, we make more statistical analysis on the data. Using the confusion matrices produced by the classifiers in the 2-fold evaluation, we calculate MaP, MaR and MaF values and report them in Table III.

TABLE III. MACRO AVERAGES OF PRECISION, RECALL AND F1 SCORES

Classifier	Type	MaP (%)	MaR (%)	MaF
SVM	Standard	95.02	89.06	91.94
DT	Standard	80.80	84.28	82.50
LR	Standard	94.15	87.13	90.50
KNN	Standard	85.48	79.00	82.11
RF	Ensemble	93.75	84.93	89.12
AdaBoost	Ensemble	94.69	88.87	91.68
MV	Ensemble	92.61	85.44	88.88
MLP	Neural Net	89.19	86.58	87.86

As we can see from the table, averaged precision, recall and F1 Scores are lower than accuracy values. The reason for this situation is that the dataset we used is an unbalanced dataset and there are some malware families that contain just a few malware samples. The tested classifiers generally do not give a good performance on these families and produce low TP rates. This situation decreases the average MaP, MaR, and MaF values. Figure 2 and Figure 3 shows the precision and recall values of each malware family based on the results of SVM classifier. Especially, FakeUpdates, Lnk and Tesbo malware families affect MaP, MaR values negatively with low precision and recall values. For some families, SVM produces high precision values but low recall values, such as Kemoge, MMarketpay, Opfake, Penetho families. This is due to low number of TP and high number of FN values in these classes. This means that SVM cannot identify the samples from these families and assign them to other families. Low number of samples in these families is the main reason of this anomaly. However, there are some malware families with low number of samples but high precision and recall values, such as Univert, Vmvol families. This shows that the extracted features provide enough information to identify these families, which probably have very different structure comparing to other families.

C. *Character Identification*

After doing classification, we also examine the features which are chosen in the feature selection step and how frequently these features are used by malware families. Based on this study we report on some characteristics of malware families and malware categories:

- Most of the malwares belonging to the families in *Adware* category, such as *Airpush*, *Kuguo* and *Youmi*, call *notify* method of *NotificationManager* class to send notification to the victim.
- The majority of the malwares in *Trojan-Banker* category leverage *SmsManager* class to send SMS to victims. *BankBot* and *SlemBunk* families are examples, which include malwares in this category.
- *SqliteDatabase* class that exposes methods to manage a SQLite database is used by most of the malwares in *Trojan-Dropper* category. However, only few members of *Ransom* and *Trojan-Spy* categories use this class.
- Malwares in *Adware* category generally detect victim's location via using methods of *Location* and *LocationManager* class. Thus, almost all of the samples in *Airpush* family call the methods of *Address* class such as *getCountryCode*, *getCountryName* and *getPostalCode*.
- Most of the malwares belonging to the *Backdoor* and *Trojan* category use the methods of *TelephonyManager* class such as *getDeviceId*, *getSubscriberId*, *getLine1Number*, and *getNetworkType* to access information about the victim's phone.
- There are many malwares in Ransom and HackerTool category whose manifest files do not include Internet permission. Particularly, Jisut family consists of mostly offline applications.
- The majority of the malwares in Ransom category request read_call_log, read_contacts and write_contacts permissions.

Although these observed behaviors are helpful for analyzing malwares, these are not certain behavior of the malware families. As the malwares evolve and new malwares are produced, these kind of behaviors will obviously change.

D. *Classification Results On Drebin and UpDroid Datasets*

To validate the classification performance of our framework, we also use Drebin and UpDroid datasets. We analyze 5507 Drebin malwares belonging to 132 different familes and 2408 UpDroid malwares belonging to 20 different families. Training and test sets are created as in the 2-fold evaluation method of AMD dataset. Additionally, we use the same feature set and same hyper-parameters which are given in Section III. Table IV shows the evaluation results of each classifiers on two datasets. Due to space limit, we just give accuracy results. MaP, MaR and MaF values can be seen in our thesis [31].

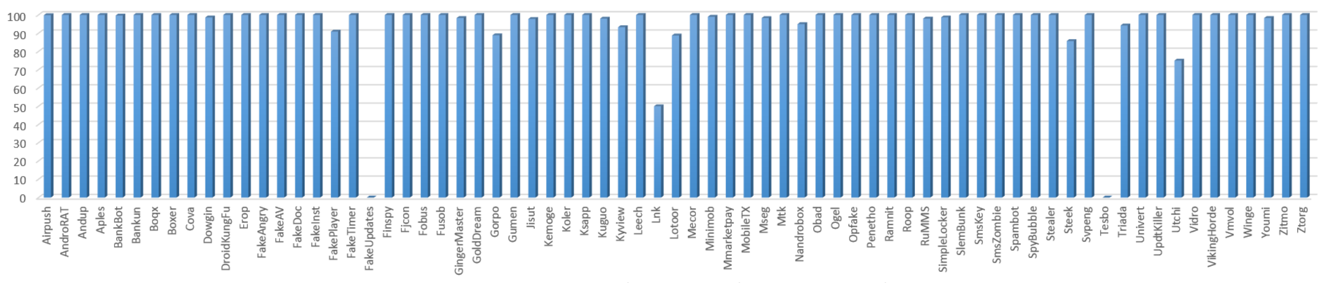

Figure 2: Precision values of SVM classifier per malware family

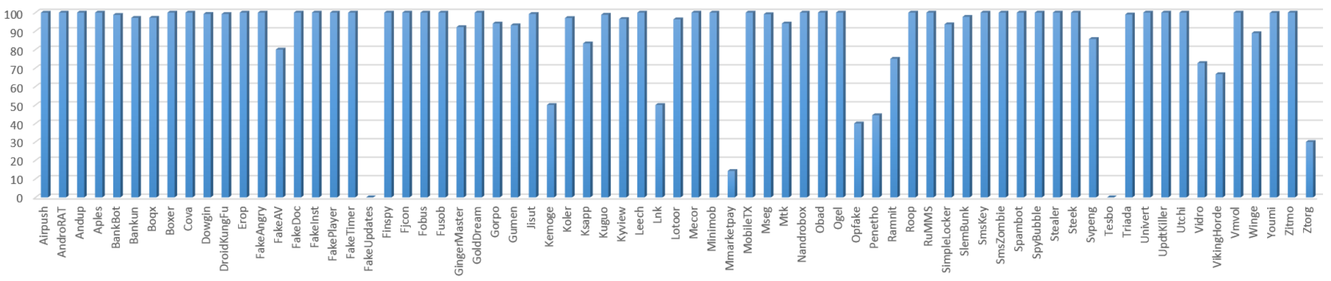

Figure 3: Recall values of SVM classifier per malware family

TABLE IV. CLASSIFICATION RESULTS ON DREBIN AND UPDROID DATASETS

Classifier	Type	Drebin - Acc (%)	UpDroid – Acc (%)
SVM	Standard	96.66	94.66
DT	Standard	92.93	90.83
LR	Standard	96.51	93.24
KNN	Standard	91.59	91.99
RF	Ensemble	96.37	94.25
AdaBoost	Ensemble	96.79	93.74
MV	Ensemble	94.84	92.16
MLP	Neural Net	96.10	93.66

Results show that all classifiers classify malwares with a high accuracy. SVM and AdaBoost classifiers give better accuracy than others as in the previous evaluation. The reason why the accuracy values are lower than the results of previous evaluation is that these datasets are smaller than AMD dataset and low number of samples lead to lower values because of the lack of training data. The results also show that the feature set obtained in AMD dataset training is useful to classify malware in another dataset.

We compare these accuracy results with the results of DroidSieve and UpDroid works. DroidSieve and UpDroid reach 97,68% and 96.85% accuracy values respectively in classification of Drebin malwares into their families. While they eliminate the most of the malware families which has low number of samples, we eliminate only the families which have just one sample. So, it is normal that we achieve lower accuracy values because of the more unbalanced dataset. UpDroid also classifies the malwares in UpDroid dataset with an accuracy of 96.2%. Our best accuracy result on this dataset is lower than this value. The reason of this difference can be the impact of dynamic analysis which is done by UpDroid.

E. Prediction on Unknown Malwares

In previous experiments, our target was classifying the malware that belongs to a known malware family. Although prediction of a malware family is an important task, detecting a new malware that doesn't belong any known family is also important in context of malware analysis. Thus, we test our model to predict unknown malwares as a final step. We leverage the probability values generated by the classifier during the classification.

As the first step, we identify the families of AMD and Drebin datasets which contain less than 25 samples and separate them as unknown malware samples. Then, we randomly select 80% of the other malwares as training set and the remaining as test set. We use SVM classifier which gives one of the best performances in the previous evaluations. After we train the classifier with training sets and test it with test sets, we note the probability of each prediction. We define a confidence value using these probabilities as shown below:

$$\text{Confidence} = \mu_p - \sigma_p$$

μ_p and σ_p denote the average and standard deviation of probability values generated during classification of all malware samples in the datasets. Using this confidence value, we try to predict an unknown malware as unknown. If the prediction probability of a given sample is lower than this value, the sample is classified as unknown. According to this model, we add the unknown samples to test sets and repeat the classification process with this dataset. Table V shows the confidence values and the classification results for each dataset. More detailed results and the impact of confidence values to these results are given in [31]. The results show that our approach is capable of detecting unknown malwares with a high accuracy.

TABLE V. CLASSIFICATION RESULTS ON UNKNOWN MALWARES

Dataset	Confidence Value	Acc (%)	Precision (%)	Recall (%)	F1-Score (%)
AMD	91.7	93.63	60.54	87.19	71.46
Drebin	85.14	86.54	86.93	84.02	85.45

V. CONCLUSIONS AND FUTURE WORK

In this paper, we propose a framework to classify Android malwares into their families. The proposed approach leverages static features and classify malwares with several machine learning classifiers. Overall, the tested machine learning models produce high accuracy in classifying malware families, which shows the saliency of the extracted features. The experimental results show that SVM classifier leads to a better accuracy among all classifiers. Since the framework is developed in a modular structure, feature extraction/selection, and machine learning techniques can be replaced with future techniques. An important conclusion is that the feature set obtained in AMD dataset is useful for classifying malwares in Drebin and UpDroid datasets. This shows the effectiveness of the proposed method in malware family classification. Furthermore, the framework can also be used to identify unknown malware samples by using probabilities generated by the classification algorithm.

As the software advances, new kind of static and dynamic analysis techniques will be needed. In the future work, the set of features can be expanded with new static and dynamic features. A future direction can be investigating unbalanced dataset problems. In most malware datasets, some of the malware families have insufficient number of samples, which makes classification of these families harder. Creating better classification techniques for unbalanced dataset would be a good contribution to the field. Additionally, a historical malware dataset can be created for analyzing evolution of malwares with time. Structures and behaviors of malwares are continuously changing. Thus, analyzing these changes would be helpful to understand future malwares.

REFERENCES

[1] Global market share held by the leading smartphone operating systems in sales to end users from 1st quarter 2009 to 2nd quarter 2018, Available at: https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/, (Accessed: Dec 16, 2018)

[2] More than 99 percent of all malware designed for mobile devices targets Android devices, explained Olaf Pursche, Head of Communications at AV-TEST, in the F-Secure State of Cyber Security 2017.

[3] AMD dataset, Available at: http://amd.arguslab.org/, (Accessed: Dec 16, 2018)

[4] Apktool, Available at: https://github.com/iBotPeaches/Apktool/, (Accessed: Dec 16, 2018)

[5] Smali, Available at: https://github.com/JesusFreke/smali, (Accessed: Dec 16, 2018)

[6] A. Feizollah, N. B. Anuar, R. Salleh, and A. W. A. Wahab, "A review on feature selection in mobile malware detection," Digital Investigation, vol. 13, no. 0, pp. 22 – 37, 2015.

[7] Wei, Fengguo & Li, Yuping & Roy, Sankardas & Ou, Xinming & Zhou, Wu. (2017). Deep Ground Truth Analysis of Current Android Malware. 252-276. 10.1007/978-3-319-60876-1_12.

[8] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In NDSS, San Diego,CA,USA, 2014.

[9] Y. Aafer, W. Du, and H. Yin. DroidAPIMiner: Mining API-level features for robust malware detection in android. In SecureComm, pages 86–103, Sydney,AU, 2013. Springer.

[10] W. Yang, X. Xiao, B. Andow, S. Li, T. Xie, and W. Enck. AppContext: Differentiating malicious and benign mobile app behaviors using context. In Proc. ICSE, pages 303–313, 2015.

[11] Hossein Fereidooni, Mauro Conti, Danfeng Yao, and Alessandro Sperduti. Anastasia: Android malware detection using static analysis of applications. In New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on, pages 1–5. IEEE, 2016.

[12] B. Amos, H. A. Turner, and J. White, "Applying Machine Learning Classifiers to Dynamic Android Malware Detection at Scale," in International Conference on Wireless Communications and Mobile Computing (IWCMC), 2013.

[13] G. Dini, F. Martinelli, A. Saracino, and D. Sgandurra. MADAM: A multi-level anomaly detector for android malware. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7531 LNCS:240–253, 2012.

[14] W.-C. Wu and S.-H. Hung. Droiddolphin: A dynamic android malware detection framework using big data and machine learning. In Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, RACS '14, pages 247–252, New York, NY, USA, 2014. ACM.

[15] Caviglione, L., Gaggero, M., Lalande, J., Mazurczyk, W., Urbanski, M.: Seeing the Unseen: Revealing Mobile Malware Hidden Communications Via Energy Consumption and Artificial Intelligence. IEEE Trans. Inform. Forensic Secur. 11, 799-810 (2016)

[16] Nancy, Dr. Deepak Kumar Sharma, "Android Malware Detection using Decision Trees and Network Traffic", in IJCSIT, Vol. 7 (4), 2016.

[17] M. Lindorfer, M. Neugschwandtner, and C. Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. In IEEE COMPSAC, 2015.

[18] F. Yang, Y. Zhuang, J. Wang, "Android Malware Detection Using Hybrid Analysis and Machine Learning Technique", from book Cloud Computing and Security: Third International Conference, ICCCS 2017, Nanjing, China, June 16-18, 2017, Revised Selected Papers, Part II (pp.565-575)

[19] L. Xu, D. Zhang, N. Jayasena, and J. Cavazos, "Hadm: Hybrid analysis for detection of malware."

[20] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, Droid-sec: Deep learning in Android malware detection, in Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication (SIGCOMM, poster), 2014, pp. 371–372.

[21] Xin Su et al. 2016. A Deep Learning Approach to Android Malware Feature Learning and Detection. In IEEE TrustCom. 244–251.

[22] L. Deshotels, V. Notani, and A. Lakhotia, "DroidLegacy: Automated familial classification of Android malware," in ACM SIGPLAN Program Protection and Reverse Engineering Workshop, PPREW, 2014.

[23] J. Garcia, M. Hammad, B. Pedrood, A. Bagheri-Khaligh, and S. Malek, "Obfuscation-resilient, efficient, and accurate detection and family identification of android malware," Department of Computer Science, George Mason University, Tech. Rep., 2015.

[24] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "Droidsieve: Fast and accurate classification of obfuscated android malware," in Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. ACM, 2017, pp. 309–320.

[25] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "Droidscribe: Classifying android malware based on runtime behavior," in Security and Privacy Workshops (SPW), 2016 IEEE. IEEE, 2016, pp. 252–261.

[26] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, R.Baldoni, "Android Malware Family Classification Based on Resource Consumption over Time".

[27] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. Blasco, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," Expert Systems with Applications, vol. 41, no. 1, pp. 1104–1117, 2014.

[28] G. Canfora, F. Martinelli, F. Mercaldo, V. Nardone, A. Santone and C. A. Visaggio, "LEILA: formaL tool for idEntifying mobIle maLicious behAviour," in IEEE Transactions on Software Engineering.

[29] T. Chakraborty, F. Pierazzi and V. S. Subrahmanian, "EC2: Ensemble Clustering and Classification for Predicting Android Malware Families," in IEEE Transactions on Dependable and Secure Computing.

[30] Scikit-learn, Available at: https://github.com/scikit-learn/, (Accessed: Dec 16, 2018

[31] Sercan Türker, Android Malware Family Classification with Machine Learning, (unpublished MSc thesis, University of Hacettepe, 2019)

[32] Aktas, Kursat & Sen, Sevil. (2018). UpDroid: Updated Android Malware and Its Familial Classification.

[33] UpDroid dataset, Available at: https://wise.cs.hacettepe.edu.tr/projects/updroid/dataset/, (Accessed: June 13, 2019)