Ensemble Machine Learning Approach for Android Malware Classification Using Hybrid Features

Abdurrahman Pektaş and Tankut Acarman(✉)

Computer Engineering Department, Galatasaray University,
Ortaköy, 34349 İstanbul, Turkey
apektas@yandex.com, tacarman@gsu.edu.tr
http://gsu.edu.tr

Abstract. Feature-based learning plays a crucial role at building and sustaining the security. Determination of a software based on its extracted features whether a benign or malign process, and particularly classification into a correct malware family improves the security of the operating system and protects critical user's information. In this paper, we present a novel hybrid feature-based classification system for Android malware samples. Static features such as permissions requested by mobile applications, hidden payload, and dynamic features such as API calls, installed services, network connections are extracted for classification. We apply machine learning and evaluate the level in classification accuracy of different classifiers by extracting Android malware features using a fairly large set of 3339 samples belonging to 20 malware families. The evaluation study has been scalable with 5 guest machines and took 8 days of processing. The testing accuracy is reached at 92%.

Keywords: Malware · Classification · Feature · Ensemble machine learning

1 Introduction

Mobile is rising with all aspects including high-end phones and tablets with powerful processors and Internet high-bandwidth connectivity. From the perspective of end users, mobile applications and services request different authorisations for access to a set of resources like database, preference, files in Android operating system (OS). In consequence, a malware may be deployed to perform malicious activities while using the privileges granted by Android OS. Android market share was 83,3% and 1.4 billion Android smartphones were sold in 2015, [16]. The growing market in smartphones and diversity in mobile applications has lead to smartphones becoming attractive target for online criminals. Again in 2015, 3.3 million applications were identified as malware targeting to steal valuable personal information. Malware writers can simply release variant of a malware

M. Kurzynski et al. (eds.), *Proceedings of the 10th International Conference on Computer Recognition Systems CORES 2017*, Advances in Intelligent Systems and Computing 578, DOI 10.1007/978-3-319-59162-9_20

by using obfuscation techniques. But since malware features are unique and variant of a malware belongs to a specific class, learning based malware classification subject to a large volume of samples is crucial for enhancing the OS security and protecting safety-critical user's information. A large variety of information-rich features is used for accurate classification of a malware sample and depending on the variety and volume of a sample set, different feature extraction and learning, decision algorithms are studied. A survey on Android security is presented in [7]. Although signature based security solutions, or namely static analysis based methods, are vulnerable to obfuscation, behavior based and dynamic analysis solutions are more reliable and accurate at detection and classification of a malware variant. In [6], N-gram searching over the static code of malware samples is used to create a classifier and a learning database is developed to determine whether an application is a malware or benign. In [12] system functions such as API calls and permissions are discovered by static analysis from applications' profile and then, classification of a sample into a malware or goodware is evaluated with a set of 1200 malware and 1200 benign samples. Permissions and events requested by Android applications are monitored, clustering and classification by K-means algorithm and a decision tree learning algorithm is studied with a set 500 sample Android applications in [4]. One-Class Support Vector Machine is trained by using the extracted permissions and a control flow graph of the input applications in [13]. The One-Class Support Vector Machine was adequate to reach at the level 80% in accuracy due to a larger set of 2081 benign sample in comparison to 91 malicious Android applications used for training. A set of 1000 malware samples belonging to 49 families and 1000 benign applications is statically analysed in [10]. Then, the Bayesian-based classifier is developed by using the features extracted from this set of known malware and benign Android applications. In [15], text mining and information retrieval is applied to a dataset of Android OS malware families for discovering similarity between those samples and an automated system is deployed to classify malware samples. In [17], a platform deployment and testing is presented. The API calls and event flows of users are collected and 15 types of features are extracted. Support Vector Machine (SVM), Decision Tree (DTree) and Random Forest algorithm is applied for classification of 666 Android applications. In our study, we apply machine learning and evaluate the level in classification accuracy of different classifiers by using the extracted Android malware features from 20 malware families with a fairly large set of 3339 samples. The analysis platform is scalable with 5 guest machines and analysis study took 8 days of processing. The testing accuracy is reached at 92%. The paper is organized as follows: In Sect. 2, we propose a novel hybrid feature-based classification system for Android malware. In Sect. 3, we define the dataset, malware families and features. Then, we apply and evaluate online machine learning algorithm to classify malware samples subject to given performance metrics. Finally, some conclusions are given.

2 Methodology

In this section, we propose a novel hybrid feature-based, i.e., static and dynamic features, classification system for Android malware. The proposed system considers static features, including permissions, contains embedded APK and hidden payload, etc. and dynamic features, including API calls, installed services, network connections to model android applications. The proposed methodology, as shown in Fig. 1, consists of five major steps. The first step is dynamic analysis of Android malware samples by Cuckoo Framework. In this step, cuckoo run samples on a virtualized environment and report their activities and some static features of the samples for example, requested permission list extracted from AndroidManifest.xml file in APK file. The second step consists of extracting appropriate features from analysis reports. In this step, we also choose some text-based features to characterize application such as methods (API calls), permissions and HTTP connections. The entire feature set is given in Table 1. Since these features need to be pre-processed before the stage of machine learning, the bag of words approach is used along with normalizing features by the means of Tf-idf transformer. After this pre-processing stage, textual features are represented with sparse matrix. In the next step, feature selection is applied on sparse matrix. We use a meta-transformer (specifically SelectFromModel in scikit-learn) along with Extra Trees Classifier [8] to select the best subset of features from the malware dataset. As a result of this process, each application is represented as feature matrix and a class label indicating Android application family. The fourth part includes the building classification model based on feature set. As Android applications are vectorised into a sparse matrix, one can simply feed this matrix into any machine learning algorithm to derive classification model. However, in our experiments, we examine three classification methods, which are more suitable to high dimensional feature space, including Logistic Regression, Naive Bayes and meta-classifier Ran-

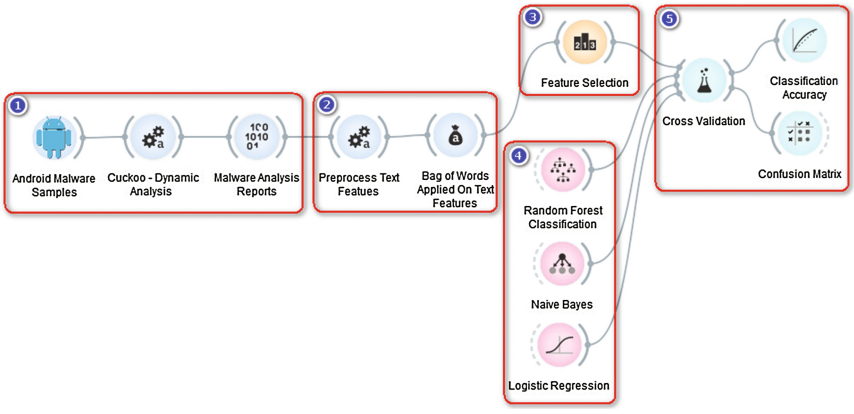

Fig. 1. Overview of the proposed methodology

Table 1. Features and their types for malware with MD5 value equals f6b60dfdab6558e26f0e92972573d0c8

Feature category	Type	Value
installed service count	Integer	5
fingerprint	String	getSimCountryIso, getDeviceId, getSimOperatorName, getLine1Number
methods	String	isVisibleToUser, getFrameTime, performAction, ...
permissions	String	Internet, Access_Network_State, Read_Phone_State, Get_Account
HTTP connections	String	http://d.applovin.com http://houseads.eu
send_sms	Boolean	False
receive_sms	Boolean	False
read_sms	Boolean	False
call_phone	Boolean	False
app_execute_shell_commands	Boolean	True
app_queried_account_info	Boolean	True
app_queried_installed_apps	Boolean	False
app_queried_phone_number	Boolean	True
app_queried_private_info	Boolean	False
app_recording_audio	Boolean	False
app_registered_receiver_runtime	Boolean	True
app_uses_location	Boolean	False
embedded_apk	Boolean	False
is_dynamic_code	Boolean	True
hidden_payload	Boolean	False
is_native_code	Boolean	False
is_reflection_code	Boolean	True

dom Forest Classification. In Sect. 3, we briefly describe these algorithms and elaborate the performance metrics, then plot confusion matrix for better illustration of levels in class-wise classification accuracy.

3 Experiments

In this section, we elaborate the prediction accuracy of algorithms subject to the feature set. Our aim is to demonstrate that combining static and dynamic features of Android application can improve classification accuracy. For this purpose, we evaluate our approach on real world malware samples. Classification

experiments are carried out on a 2.5GHz Intel 4-Core i-7 processor with 8GB physical memory, using scikit learn [1,11] and MS Windows 10.

3.1 Dataset

The benchmark malware dataset is obtained from "VirusShare Malware Sharing Platform" [2]. This platform provides up-to-date malware samples from various types including windows executable (exe, dll), javascript, Android samples (APK), Java, PDF. The Ubuntu 16.04 desktop operating system installed on a hardware Intel(R) Core(TM) i5-2410M@2.30 GHz processor and 2 GB of memory is used to conduct the experimental study. The analysis was scalable with 5 guest machines and took 8 days of processing 3339 samples. To label malware samples, Virustotal as an online web-based multi anti-virus scanner is used [3]. The malware classes and class-specific measures are written in Table 2.

Table 2. Malware families and their class-specific measures

Class	Code	Count	Precision	Recall	F1-score
android.adware.adwo	0	97	0.75	0.82	0.78
android.adware.appquanta	1	70	1.00	0.67	0.80
android.adware.dowgin	2	867	0.86	0.99	0.92
android.adware.gingermaster	3	89	0.83	0.62	0.71
android.adware.kuguo	4	63	1.00	0.25	0.40
android.adware.plankton	5	42	1.00	1.00	1.00
android.adware.utchi	6	45	1.00	1.00	1.00
android.adware.wapsx	7	127	0.75	0.82	0.78
android.adware.youmi	8	125	0.86	0.75	0.80
android.exploit.gingerbreak	9	37	0.50	1.00	0.67
android.exploit.psn	10	41	0.00	0.00	0.00
android.riskware.agent	11	93	0.50	0.10	0.17
android.riskware.smspay	12	311	0.82	0.88	0.85
android.riskware.smsreg	13	133	0.73	0.47	0.57
android.trojan.agent	14	84	0.70	0.78	0.74
android.trojan.clicker	15	50	1.00	1.00	1.00
android.trojan.fakeinst	16	207	0.96	1.00	0.98
android.trojan.smskey	17	75	0.67	0.33	0.44
android.trojan.smssend	18	81	0.88	0.70	0.78
trojan.java.smssend	19	702	0.81	1.00	0.90
Average			0.82	0.84	0.81

3.2 Evaluation Metrics

The proposed classification method is evaluated by using the following metrics: ***precision***, ***recall*** (a.k.a. sensitivity), ***F1-score***, ***classification accuracy*** (the overall correctness of the model). In binary classification (positive and negative classes), true positives (*tp*) refer to the correctly predicted positive samples, while true negatives (*tn*) are the number of the correctly predicted negative samples. False positives (*fp*) refer to the incorrectly classified positive samples. Similarly, false negatives (*fn*) are the number of incorrectly classified negative samples. On one hand, the terms positive and negative denote the classifier's success, on the other hand true and false determines whether or not the prediction is matched with the actual (i.e., ground truth) label. The precision is the proportion of the sum of true positives versus the sum of positive instances. For instance, it is the probability for a positive sample to be classified correctly. The recall is the proportion of instances that are predicted positive and are also actually positive (i.e., *tp*) of all the instances that are positive. The F1-score, also known as F-measure or F-score, is the weighted harmonic mean of the precision and recall. F1-score reaches at its best value 1 and the worst score at 0. In binary classification problem, the precision and recall contribute equally to F1-score. However, in the multi-class problems, overall F1-score is calculated by taking the weighted mean of the F1-score of each class. The F-score is a popular measure used in the natural language processing tasks. The metrics are given as follows:

$$precision = \frac{tp}{tp\ +\ fp} \tag{1}$$

$$recall = \frac{tp}{tp\ +\ fn} \tag{2}$$

$$F1\ -\ score = 2 \times \frac{precision \times recall}{precison + recall} \tag{3}$$

$$accuracy = \frac{correctly\ classified\ instances}{total\ number\ of\ instances} \tag{4}$$

3.3 Classification Methods

We evaluate different classifiers, including logistic regression, naive bayes, random forest. The objective is to assess whether combining static and dynamic feature can provide sufficient information in describing Android application. Additionally, we also aim to determine the best classification method in terms of classification accuracy. In machine learning, multinomial Logistic Regression algorithm ([14,18]) is a classification technique that learns and fits the data based on logit function (or logistic curve). In order words, multinomial logistic regression is aimed to predict the outcomes of a multi-class problem according to given a set of feature including real-valued, binary-valued or categorical-valued. It is capable of dealing with a large number of features. The Multinomial Naive Bayes (MNB) classifier ([9]) is a variation of the probabilistic classification algorithm based on

the Bayes' Rule. The algorithm assumes that the attributes, namely features, are independent from each other given the class. MNB classifier is highly scalable method for high dimensional dataset and also it takes linear time to make a prediction. As a result, it is a suitable method for text classification problem such as spam detection, information retrieval and text categorisation. Random Forest ([5]) is an ensemble method designed to increase the accuracy of the decision tree by using collection of decision tree. Each tree is trained on randomly selected features, and each tree votes for the most popular class. Then, the output of the classifier is determined by integrating the votes of trees. Consequently, the Random Forest algorithm can handle high dimensional feature space while being computationally less expensive when compared to other ensemble methods. Moreover, employing a set of trees leads a significant increase in classification accuracy.

3.4 Results

We used 10-fold cross-validation approach to measure accuracy. However, as our dataset is imbalanced in terms of the number of samples in each class, we adopted Stratified K-Folds method through the evaluation process. Stratified K-Folds validator splits the data into train and test sets by preserving the percentage of the samples for each class. Table 3 shows general classification accuracy and average recall, precision and F1-score for the tested machine learning algorithm. According to our evaluation, meta-classifier Random Forest outperforms the other two classifiers and achieves the highest accuracy with 84%. We also consider a base class that is constituted by trojan, riskware and adware targeting Android OS. This base class is primarily the largest set of variants that need to be identified at day-0. The classification accuracy of Random Forest applied to the base class is reached at 92%. The accuracy of the classifier at recognising instances of different classes is illustrated with the confusion matrix as plotted in Fig. 2. The confusion matrix compares the number of correct and incorrect predictions of the classifier with respect to the ground truth (actual classes). From confusion matrix, it can be seen that the model is successful in determining the majority of the malware families. However, one can remark that, android.exploit.psn (numbered as 10 for the sake of appearance) family was always wrongly classified as trojan.java.smssend (numbered as 19). Another remark is that due to the usage of imbalanced dataset, the classifier tends to classified malware samples

Table 3. Classification accuracies of the tested machine learning algorithms subject to all classes and base class

Algorithm	Accuracy		Precision		Recall		F1-score	
	All classes	Base classes	All	Base	All	Base	All	Base
Random forest	0.84	0.92	0.82	0.92	0.84	0.92	0.81	0.92
Logistic regression	0.81	0.91	0.81	0.91	0.81	0.91	0.78	0.91
Naive Bayes	0.64	0.83	0.53	0.83	0.64	0.82	0.53	0.82

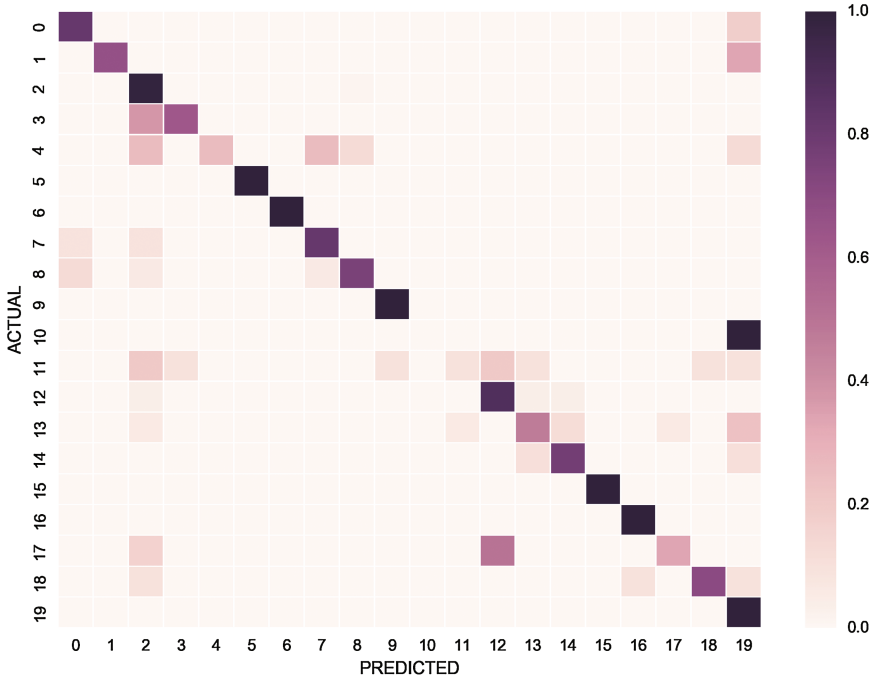

Fig. 2. Normalized confusion matrix for all classes

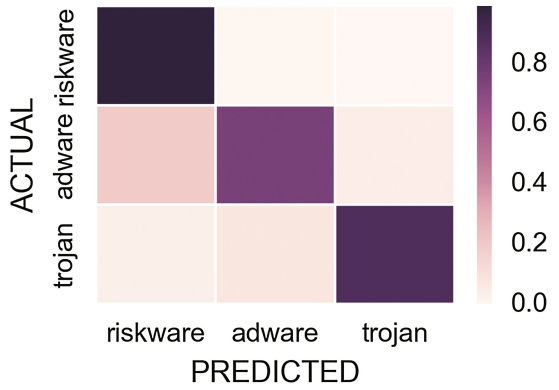

Fig. 3. Normalized confusion matrix for base classes

into android.adware.dowgin and trojan.java.smssend families, which are the first two classes that contain the most members in our dataset. Figure 3 shows the class-wise classification accuracy for the base classes, and the model is capable of predicting each base class with high accuracy.

4 Conclusions

In this paper, we propose a novel classification method for Android malware samples according to their static and dynamic features (i.e., hybrid features). The proposed framework executes the given samples in a virtualized environment and extracts its activities such as API calls, network connections, and so on. Then, that information is further combined with static features particularly with a permission list provided by the application. By applying learning algorithms to the benchmark datasets, the Random Forest algorithm achieved the highest accuracy at 84% while classifying applications into their respective families. Our experiments on real-world Android malware samples verify the contributions of the proposed method at identifying the Android families on a basis of their extracted hybrid features.

Acknowledgements. The authors gratefully acknowledge the support of Galatasaray University, scientific research support program under grant #16.401.004.

References

1. Scikit-learn: Machine Learning in Python. http://scikit-learn.org/stable/index.html. Accessed 15 Jan 2017
2. Virusshare: Malware Sharing Platform. https://virusshare.com/. Accessed 15 Jan 2017
3. Virustotal: Free Online Virus, Malware and URL Scanner. https://www.virustotal.com/. Accessed 15 Jan 2017
4. Aung, Z., Zaw, W.: Permission-based android malware detection. Int. J. Sci. Technol. Res. **2**, 228–234 (2013)
5. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
6. Dhaya, R., Poongodi, M.: Detecting software vulnerabilities in android using static analysis. In: 2014 IEEE International Conference on Advanced Communications, Control and Computing Technologies, pp. 915–918, May 2014
7. Faruki, P., Bharmal, A., Laxmi, V., Ganmoor, V., Gaur, M.S., Conti, M., Rajarajan, M.: Android security: a survey of issues, malware penetration, and defenses. IEEE Commun. Surv. Tutorials **17**(2), 998–1022 (2015). (Secondquarter)
8. Geurts, P., Ernst, D., Wehenkel, L.: Extremely randomized trees. Mach. Learn. **63**(1), 3–42 (2006)
9. John, G.H., Langley, P.: Estimating continuous distributions in Bayesian classifiers. In: Proceedings of the Eleventh conference on Uncertainty in artificial intelligence, pp. 338–345. Morgan Kaufmann Publishers Inc. (1995)
10. McWilliams, G.: Analysis of Bayesian classification-based approaches for android malware detection. IET Inf. Secur. **8**(1), 25–36 (2014). http://digital-library.theiet.org/content/journals/10.1049/iet-ifs.2013.0095
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in Python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
12. Peiravian, N., Zhu, X.: Machine learning for android malware detection using permission and API calls. In: Proceedings of the 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, ICTAI 2013, pp. 300–305 (2013). http://dx.doi.org/10.1109/ICTAI.2013.53

13. Sahs, J., Khan, L.: A machine learning approach to android malware detection. In: 2012 European Intelligence and Security Informatics Conference, pp. 141–147, August 2012
14. Schmidt, M., Le Roux, N., Bach, F.: Minimizing finite sums with the stochastic average gradient. Math. Program. **162**(1), 83–112 (2017). http://dx.doi.org/10.1007/s10107-016-1030-6
15. Suarez-Tangil, G., Tapiador, J.E., Peris-Lopez, P., Blasco, J.: Dendroid: a text mining approach to analyzing and classifying code structures in android malware families. Expert Syst. Appl. **41**(4), 1104–1117 (2014). http://dx.doi.org/10.1016/j.eswa.2013.07.106
16. Symantec: Internet security threat report (2016). https://www.symantec.com/content-/dam/symantec/docs/reports/istr-21-2016-en.pdf
17. Yang, Y., Wei, Z., Xu, Y., He, H., Wang, W.: Droidward: an effective dynamic analysis method for vetting android applications. Cluster Comput. **19**, 1–11 (2016)
18. Yu, H.F., Huang, F.L., Lin, C.J.: Dual coordinate descent methods for logistic regression and maximum entropy models. Mach. Learn. **85**(1–2), 41–75 (2011)