JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

A Study on Android Malware Detection Using Machine Learning

[1] Kotturthi Renuka, [2] Dr.K.RajKumar
[1]M.tech, Department of Computer Science
Andhra University, Visakhapatnam, AP, India
[2]Asst. Professor, Department of Computer Science,
Andhra University, Visakhapatnam, AP, India.

ABSTRACT: Smart phones are becoming essential in our lives, and Android is one of the most popular operating systems. Android OS is wide-ranging in the mobile industry today because of its open source architecture. It is a wide variety of applications and basic features. App users tend to trust Android OS to secure data, but it has been shown that Android is more vulnerable and unstable. Identification of Android OS malware has become an emerging research subject of concern. This paper aims to analyse the various characteristics involved in malware detection. Mobile devices have grown exponentially in terms of functionality in the recent years, since they provide almost all the functions that a computer provides. Among the various operating systems employed, Android has become a prominent one in the recent years due to its huge user base, since the availability of android applications is free in the android application market.

Due to his huge user base it has become a very likely target for attackers. Among the available applications a vast majority of them are not authenticated formally and hence are malicious. These applications may steal the private information from the user's device. The proposed framework ensures that these kind of applications are detected at high accuracy, it provides a machine learning-based malware detection system on Android to detect the malicious applications to enhance security and privacy of smartphone users. The proposed framework monitors various permissions related to the android applications and analyses the features by using machine learning classifiers to authenticate the applications. It also addresses malware detection methods. The current detection mechanism utilizes algorithms such as SVM, Neural Networks and other algorithms for machine learning to train the sets and find the malware. The results of our empirical evaluation show our system is competitive in terms of classification accuracy and detection efficiency. At dataset Drebin (benign 5.9K and malware 5.6K) and AMD (benign 20.5K and malware 20.8K), our system has achieved 96% and 98% detection results both in accuracy and F-measure. Compared with the state-of-the-art system in detecting evolving malware called MaMaDroid on the dataset of 6.0K benign and 20.5K malicious samples spanning from 2010 to 2017, our system achieves higher accuracy while improving detection efficiency by 15 times..

KEYWORDS*:* APks Dataset, SVM algorithms, MLP, Malware detection.

I. INTRODUCTION

Now-a-days the role of mobile phones in the human lives has increased. Android has become an inseparable part of the current mobile systems, due to the openness of free source. Android covers around 85% of world's smartphone market until 2018. At the same time, the open source availability is also a bait since it attracts a lot of attackers. According to the recent report, in 2018, 360 Internet Security Center intercepted about 4.342 million new malware on mobile terminals, or about 12,000 per day. These malicious apps are created to perform different types of attacks such as stealing private information, sending message without the user permission, baiting users to malicious websites, etc., which may be serious threat to the privacy of users. By time these malicious applications have become hard to detect or even prone to detection, and even some malicious apps have more than 50 variants, making it hard for detection. Therefore, the detection techniques must be rationalized to detect these types of malwares in every circumstances. The researches based on permissions and intents of Android apps are more prone to false positives, since benign apps also require sensitive permissions, which make them to be misclassified as malware easily. An Android malware detection method based on method-level correlation relationship of application's API calls is proposed. The behaviour of an application is determined by the source code through the user-defined methods, and each of the methods implement specific operations by invoking API calls. The process of differentiating the combinations of API calls in the method of

malicious and benign apps is the key to establish the detection system. Therefore, association rule is introduced to analyse technology and characterize the API calls' relationships in the same method and capture app's behavioural information.
In the implementation of Android malware detection using machine learning, the two primary sources of the feature are static extraction and dynamic extraction. Static features are extracted from the manifest, Dalvik bytecode, native code, sound, image, and other reversed APK files. Dynamic features are collected from the log records, code execution paths, variable value tracking, sensitive function calls, and other behaviours in the process of application execution by running APK files in a monitored environment.
Although the detection method based on static features has some limitations compared with those based on dynamic ones, such as it is challenging to combat code obfuscation, it also has distinct advantages:(i)Full code coverage: static feature extraction can cover all code and all resource files by scanning code or symbolic pseudo execution. In contrast, dynamic feature extraction can hardly cover all code execution paths. Many applications require users to provide login credentials to use most of the features, making it difficult to detect all the functions in dynamic execution, resulting in incomplete feature extraction.(ii)Reliable detection efficiency: static feature extraction will complete the detection task in the expected time because it does not need to run the application. In contrast, dynamic feature extraction requires triggering various functions in code execution, which will consume lots of time. While the application is running, it takes some time to simulate a click-through interface. The program may perform a very complex computation or enter an infinite loop. These conditions make it difficult for the detection task to be completed within the specified time frame.(iii)Unperceived by malicious code: static detection does not require the codes' execution, so malicious applications cannot recognize that they are under check. Although some malware attempt to make the static analysis more challenging by setting up interference codes, these added codes may themselves be an identifier to assist in identifying malicious applications.(iv)Easier to generate generic fingerprint identification: static malicious sample analysis is more inclined to extract features with invariance and universality. In contrast, dynamic analysis is very likely to be affected by the operating environment. Statically extracted features are suitable for fingerprinting and can be used for the rapid predilection of large-scale malicious applications.

There are some surveys about Android malware detection published in the past few years. The authors in analysed the Android security mechanism and typical malware detection methods. The authors in focused on applying deep learning algorithms such as Restricted Boltzmann Machines, Convolutional Neural Network, Deep Belief Network, Recurrent Neural Network, and Deep Auto encoder to malware detection and analysed the advantages and results achieved. The authors in is mainly concerned with the analysis of Android malware variants' detection methods. The authors in investigated Android malware detection and protection technology based on data mining algorithms. The study in collected the literature research of the past few years, systematically analysed the static detection technology, and discussed datasets, features, algorithms, empirical experiments, and performance measures. The study in introduced the Android architecture, security mechanism, malware classification, and entire detection process, including sample collection, data pre-processing, feature selection, machine learning model construction, and experimental evaluation. The study in comprehensively discussed static, dynamic, and hybrid detection techniques. The study in mainly focused on mobile malware detection techniques, analysed signature-based detection, anomaly-based detection, and other traditional detection methods. The study in discussed various threats and the current Android platform security state, introduced three attack types, explained the factors contributing to the increase of malware, and analysed defensive mechanisms of Android protection.

The above surveys have done excellent work, but there are still some aspects that can be improved. For example, the sources of static features, the challenges of obfuscation technology to static analysis, and the deterioration issues of machine learning models are not investigated in detail. Our work aims to provide a comprehensive survey about Android malware static detection based on machine learning technologies. To this end, we searched in IEEE, ACM, Springer, Wiley, Hindawi, and other databases and used Google Scholar and DBLP to find the related papers. It is worth noting that we only use research papers from DBLP since 2016 in statistics of static feature types, machine learning algorithms, dataset usage, papers' number, and evaluation metrics. The reason is that DBLP has become an authentic database, and the papers it saves are of relatively high quality and are relatively few in number. We can manually check the detection technology, algorithm model, and evaluation method used in each article to form accurate statistical results. Based on the papers we collected on Android static malicious application detection, we finish this survey work.

In our work, we analysed the Android application static features and the typical obfuscation methods, discussed machine learning algorithms suitable for Android malware detection, explained evaluation metrics of machine learning models and sustainability issues, investigated the technical route, advantages, and disadvantages of the existing research, and made an outlook on the possible future research directions in this field. The main contributions can be summarized as follows :(i)We carried out a comprehensive review of Android malware's various static detection methods based on machine learning. The basic principles, feature sources, datasets, performance metrics, contributions, and limitations of the methods were compared vertically.(ii)We analysed the Android application composition, the source of static feature extraction, and the feature vector generation method in detail.(iii)The limitations of current methods were discussed, and the future development directions were prospected.

2. RELATED WORK

Hanqing Zhang et. al [1] presents a system which first splits each android application's source code into function methods, and the abstracted API calls of them is formed into a set of abstracted API calls transactions, whose confidence of association rule is

calculated to form the behavioral semantics for describing an application. Further using machine learning the system can differentiate between benign and malicious applications.

Zarni Aung et. al [2] presents a method in which the features from the Android. apk files are extracted. The extracted features are added in a dataset, which forms the basis of the malware detection framework. Using machine learning approaches the validation process is done.

Pengbin Fen et. al [3] presents a dynamic analysis framework called Android, based on dynamic behavior features. Android uses ensemble learning algorithm to distinguish between malicious and benign. It also employs feature selection algorithm which removes unwanted noise and features and extracts critical behavior features.

Gianluca Dini et.al[4] differentiates malwares into different classes based on their actions. MADAM a host based malware detection system is employed which simultaneously analyses the features at different levels, such as kernel, application, user and package. MADAM employs a very huge dataset thus this system ensures safety from almost every malware

In [7], the model named FAMD (Fast Android Malware Detector) was proposed that extracted Dalvik code sequences and permissions from the samples and Applied CatBoost classifier to detect malware apps. In [8], the authors extracted various static features from source code and manifest files, and a linear SVM algorithm was applied to detect malicious apps.

Similarly the authors in [9] applied machine learning algorithms on several static features and manifest components for detection. In some other works like [10], and [11] malware detection is performed by extraction of static features on the Android platform. The identification of the most important permissions to differentiate malware from benign apps is done by the SIGPID model presented in [12]. The authors applied pruning with association rules mining for ranking and identifying the important permissions.

The dangerous and harmful patterns of permissions within the malicious apps were analyzed by Moonsamy et al. [13]. The authors in [14] combined permissions and intents for detection using PCA and machine learning techniques. The authors in [15] used permissions for malware detection using gain ratio, J48, Multilayer Perceptron, Sequential Minimal Optimization (SMO), and Randomizable filtered classifiers. Permissions and intents were ranked with information gain in [16] and further, that ranking was applied to detect malicious apps. Likewise, in [17] and [18], authors applied various machine learning algorithms on manifest Components for malware detection.

In [19], factorization machine architecture was applied by the authors on the collection of manifest features for malware detection in apps.

The malign score was further used for malware detection. Some recent techniques like [20] and [21] have also examined manifest features for malware detection. However, none of the above-mentioned works have aimed to find the best set of manifest features for effective Android malware detection. In this work, we aim to find the best set of manifest features which can give better accuracy in the detection of malicious apps.

The authors in [22] applied Factorization Machine architecture on the set of manifest components to detect malicious Android apps. Sato et al. [23] evaluated the malign score for each of the manifest file components depending upon the number of malicious applications the component is present in. They further used that malign score for malware detection.

The work done in [24] is about a Feature transformation-based Android Malware detector which takes well-known features and introduces three new types of feature transformations that transform these features irreversibly into a new feature domain. None of the above-discussed works have aimed to find the best feature set comprising manifest file components for malicious apps detection in Android. However, our work aims to find the best feature set that could give better detection accuracy.

3. PROPOSED MACHINE LEARNING MODELS

SUPPORT VECTOR MACHINE

In machine learning, support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data for classification and multivariate analysis. Developed at AT&T Bell Laboratories by Vladimir Vapnik with colleagues, SVMs are one among the foremost robust prediction methods, being supported statistical learning frameworks or VC theory proposed by Vapnik and Chervonenkis. Given a group of coaching examples, each marked as belonging to at least one of two categories, an SVM training algorithm builds a model that assigns new examples to at least one category or

the opposite , making it a non-probabilistic binary linear classifier (although methods like Platt scaling exist to use SVM during a probabilistic classification setting). An SVM maps training examples to points in space so on maximize the width of the gap between the 2 categories. New examples are then mapped into that very same space and predicted to belong to a category supported which side of the gap they fall. Type-II Diabetes has varying levels of seriousness. It usually gets worse over time though treatment has been shown to slow progression. If left untreated, Type-II Diabetes can progress to Diabetes and early cardiovascular disease..

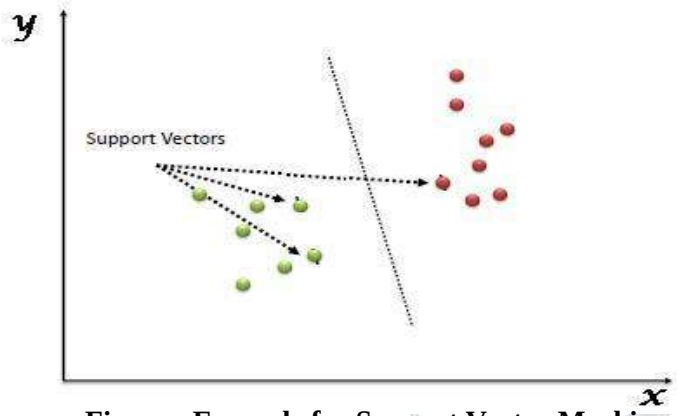
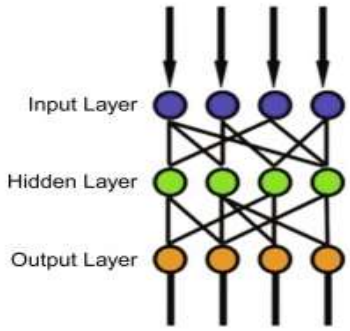


Figure : Example for Support Vector Machine

MULTI LAYER PERCEPTRON

Multi-layer perceptron (MLP) is a supplement of feed forward neural network. It consists of three types of layers—the input layer, output layer and hidden layer, as shown in Fig. 3. The input layer receives the input signal to be processed. The required task such as prediction and classification is performed by the output layer. An arbitrary number of hidden layers that are placed in between the input and output layer are the true computational engine of the MLP. Similar to a feed forward network in a MLP the data flows in the forward direction from input to output layer. The neurons in the MLP are trained with the back propagation learning algorithm. MLPs are designed to approximate any continuous function and can solve problems which are not linearly separable. The major use cases of MLP are pattern classification, recognition, prediction and approximation. For sequential data, the RNNs are the darlings because their patterns allow the network to discover dependence on the historical data, which is very useful for predictions. For data, such as images and videos, CNNs excel at extracting resource maps for classification, segmentation, among other tasks. In some cases, a CNN in the form of Conv1D / 1D is also used for networks with sequential input data. However, in most models of Deep Learning, MLP, CNN, or RNN are combined to make the most of each.

MLP, CNN, and RNN don't do everything. Much of its success comes from identifying its objective and the good choice of some parameters, such as Loss function, Optimizer, and Regularizer. We also have data from outside the training environment. The role of the Regularizer is to ensure that the trained model generalizes to new data



Classification Accuracy

Accuracy of the constructed classifier model can be calculation using the following equation.

TP+TN

Accuracy = (1)

TP+TN+FP+FN

Where,

TP = Observation is positive an predicted is also positive

TN=Observation is negative and predicted is also negative

FP = Observation is negative but predicted is positive

FN = Observation is positive but predicted is negative

4. METHODOLOGY

DESCRIPTION OF DATASET

The dataset describes the process of extracting features from the Android .apk files and to create a dataset from the extracted feature of Android applications in order to develop android malware detection framework. In pre-processing, the raw data are removed from the data set. Feature extraction is based on associated analysis of the API call's behaviour. Malware and benign usually show different behavioural patterns in the construction of function method. By using classification algorithm, the converted data set is Classified as benign and malicious The most popular R Programming Data analytics tool has been used to construct the prediction framework.

DATA PRE-PROCESSING

Dataset is collected from kaggle, Since APK files cannot be analysed directly, and some pre-processing is required before feature extraction. Unlike the application for desktop based Portable Executable (PE) files, Android app also called as APK file is in zip file, and it can be opened with unzipping tools such as WinRAR. After decompressing the APK file, the following files are obtained: AndroidManifest.xml, META-INF, res, lib, assts, clssses.dex, resources.arsc. ''classes.dex'' file is generated after compiling the code written for Android and could be interpreted by the DalvikVM. In order to get the Android app's behavioral data, there the dex file is to be converted to analyzable format. Smalli code can be decompiled directly from APK files, and contains all the needed information, thus, it becomes the target format. Thus used for creating a dataset from extracted features of Android applications in order to develop android malware detection framework. For each Android application, several selected features are retrieved from the corresponding application package (APK) file. The values of selected features are stored as a binary number (0 or 1), which is represented as a sequence of comma separated values. Each item includes the name of a feature, the data type of the feature, and data of the feature.

MANIFEST FILE EXTRACTION

Static features that we intend to use in our research work are defined inside the manifest file of Android applications. Hence, to begin with, we first extract the manifest files of the applications using Apktool. This tool breaks the application (apk package file) into several components, including its manifest file.

FEATURES EXTRACTION

The basic unit of application's behavioural semantics the method defined in the app's source code. The behavioral pattern of the malware and benign applications differs in the process of construction of the function methods, manifested in different API combinations owned by different function method. A method level associated analysis is used for the construction of the characteristics, since there is a need to discover the pattern of behaviour

FEATURE RANKING

After creating the Bag-of-Words model for each manifest component in the matrix form, we calculate the Information Gan (I.G.) score for each unique feature present in S(i) feature set. Information Gain gives the measure of the information gained about a random variable by observing another random variable. In other words, it can be used to determine, amongst a given set of features, which one is the most distinguishing, i.e., the feature that can better differentiate between several categories. Entropy defines the uncertainty in all the features and I.G. determines the reduction in that uncertainty

BAG-OF-WORDS MODEL

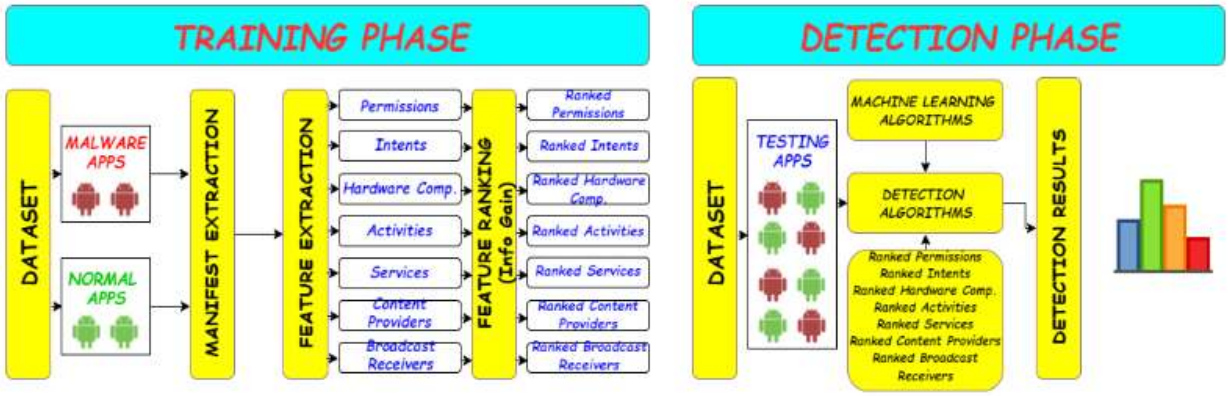
After extracting the seven features from the manifest files of all the apps, we use the Word Embedding Technique in NLP for creating the Bag-of-Words Model for each of the seven features, represented by the sets S(1), S(2), S(3),....S(7), i.e., S(i) for i ranging from 1 to 7. A bag-of-words model is just a matrix that represents the occurrence of various words that are present in a corpus or document. The current model only focuses on whether a particular word, i.e., feature in our case, occurs in the manifest file or not. The proposed methodology helps in the matrix representation of each manifest file component, which is later used by machine learning algorithms; wherein each unique entry in the feature set is represented as a column and applications as rows.

5. RESULTS

Models has been constructed using training data set (280 instances) which is 70% of original Type-II Diabetes data set. Constructed models have been validated using test data which is 30% of original data with respect to the parameter accuracy. Here, Accuracy has been calculated using confusion matrix .The best classifier model is the one with highest accuracy.

Accuracy of SVM Confusion Matrix has been generated by SVM model for the test data(120 instances) with class as the target variable. The confusion matrix clearly says that 2 instances are not classified properly and 118 instances have been classified accurately and the accuracy of this classifier model is **98.33%**

Accuracy of MLP Confusion Matrix has been generated by SB-SVM model for the test data(120 instances) with class as the target variable. The confusion matrix clearly says that 2 instances are not classified properly and 118 instances have been classified accurately and the accuracy of this classifier model is **97.43%**



.

Paste your screenshots here

CONCLUSION

This project presented a framework is developed for classifying Android applications as benign or malicious applications using machine-learning techniques. The applications are downloaded from the android application market. To generate the models, several features from these applications are extracted. Some of the malware applications are taken from malware sample database and both malware and normal applications are classified by using machine learning techniques. In order to validate the methods used,200 samples of Android applications are collected and the features are extracted for each application and the trained models will evaluate them. we proposed a static model to detect malicious Android applications by analyzing manifest file components. The proposed model aimed to rank each of the manifest component using the concept of relative frequency, i.e., by comparing its frequency in both malware and normal dataset. We constructed, for each of the manifest components, two rankings, one highlighting the features significantly present in malware apps and the other representing the features significantly present in normal apps. Thereafter, we proposed a novel algorithm to find the best set of manifest features that gives highest detection accuracy. The proposed detection algorithm applied several machine learning classifiers on the ranked list of manifest features to get their best set. We achieved the accuracy of 95.90% with the proposed model on the best set of 36 features. In our future

work, we will analyze, in addition to the manifest components, the Java source code and API calls of the apps to further improve the detection accuracy.

FUTURE WORK

Android malware detection by analyzing the manifest file components. To begin with, we ranked all the manifest file components using Information Gain. Further, we proposed a novel method that applied various machine learning and deep learning algorithms on the ranked components. The algorithm produced the best set of components that gave better accuracy as compared to any other set of components. The results highlighted that combining all the manifest file components for malware detection gave better results than the individual components. In our future work, we will put forward a novel technique that combines, with manifest file components, other features such as API calls, system calls, network traffic, etc., to detect the stealthier malware samples..

REFERENCES

[1] (2019). Smartphone Market Share. [Online]. Available http://www.michaelshell.org/tex/ieeetran

[2] (2018). 12,000 New Samples Per Day. [Online]. Available: http://blogs.360.cn/post/review_android_malware_of_2018.html?from=timeline

[3] L. Onwuzurike, E. Mariconti, P. Andriotis, E. De Cristofaro, G. Ross,and G. Stringhini, ``MaMaDroid: Detecting Android malware by buildingMarkov chains of behavioral models (extended version)," ACM Trans.Privacy Secur., vol. 22, no. 2, p. 14, 2019.

[4] Y. Aafer, W. Du, and H. Yin, ``DroidAPIminer: Mining API-level featuresfor robust malware detection in Android," in Proc. Int. Conf. Secur. PrivacyCommun. Syst. Cham, Switzerland: Springer, 2013, pp. 86_103.

[5] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, ``Drebin: Effective and explainable detection of Android malware in your pocket," in Proc. Annu. Symp. Netw. Distrib. Syst.Secur. (NDSS), vol. 14, 2014, pp. 23_26

[6] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, ‘‘Deep ground truth analysis of current Android malware,’’ in Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment. Cham, Switzerland: Springer, 2017,pp. 252–276.

[7] I. Burguera, U. Zurutuza, and S. Nadjm-Tehrani, ‘‘Crowdroid: Behavior-based malware detection system for Android,’’ in Proc. 1st ACM Workshop Secur. Privacy Smartphones Mobile Devices, 2011, pp. 15–26.

[8] W.-C. Wu and S.-H. Hung, ‘‘DroidDolphin: A dynamic android malware detection framework using big data and machine learning,’’ in Proc. Conf.Res. Adapt. Convergent Syst., 2014, pp. 247–252.

[9] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, ‘‘MADAM:Effective and efficient behavior-based Android malware detection and prevention,’’ IEEE Trans. Depend. Sec. Comput., vol. 15, no. 1, pp. 83–97, Jan./Feb. 2018.

[10] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, ‘‘The evolution of Android malware and Android analysis techniques,’’ ACM Comput. Surv., vol. 49, no. 4, p. 76, 2017.

[11] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, ‘‘DroidMat: Android malware detection through manifest and API calls tracing,’’ inProc. 7th Asia Joint Conf. Inf. Secur., Aug. 2012, pp. 62–69.

[12] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, ‘‘Significant permission identification for machine-learning-based Android malware detection,’’ IEEE Trans. Ind. Informat., vol. 14, no. 7, pp. 3216–3225, Jul. 2018.

[13] R. Kumar, X. Zhang, R. U. Khan, and A. Sharif, ‘‘Research on data mining of permission-induced risk for Android IoT devices,’’ Appl. Sci., vol. 9, no. 2, p. 277, Jan. 2019.

[14] S. Hou, Y. Ye, Y. Song, and M. Abdulhayoglu, ‘‘HinDroid: An intelligent Android malware detection system based on structured heterogeneous information network,’’ in Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining, 2017, pp. 1507–1515.

[15] A. Shabtai, Y. Fledel, and Y. Elovici, ‘‘Automated static code analysis for classifying android applications using machine learning,’’ in Proc. Int. Conf. Comput. Intell. Secur., Dec. 2010, pp. 329–333.

[16] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, ‘‘Effectiveness of opcode ngrams for detection of multi family Android malware,’’ in Proc. 10th Int. Conf. Availability, Rel. Secur., Aug. 2015,pp. 333–340.

[17] J. Zhang, Z. Qin, K. Zhang, H. Yin, and J. Zou, ‘‘Dalvik opcode graph based Android malware variants detection using global topology features,’’ IEEE Access, vol. 6, pp. 51964–51974, 2018.

[18] T. Gao, W. Peng, D. Sisodia, T. K. Saha, F. Li, and M. Al Hasan, ‘‘Android malware detection via graphlet sampling,’’ IEEE Trans. Mobile Comput.,to be published.

[19] Z. Ma, H. Ge, Y. Liu, M. Zhao, and J. Ma, ‘‘A combination method for Android malware detection based on control flow graphs and machine learning algorithms,’’ IEEE Access, vol. 7, pp. 21235–21245, 2019.

[20] A. Cutler, D. R. Cutler, and J. R. Stevens, ‘‘Random forests,’’ Mach. Learn.,
vol. 45, no. 1, pp. 157–176, 2004.

[21] Androguard. Accessed: 2019. [Online]. Available: https://github.com/androguard

[22] Winrar. Accessed: 2019. [Online]. Available: https://www.win-rar.com/

[23] Dalvik Opcodes. Accessed: 2019. [Online]. Available: http://pallergabor. uw.hu/androidblog

[24] Android APIS Reference. Accessed: 2019. [Online]. Available:
http://www.android/doc.com/reference/packages.html

[25] R. Agrawal, T. Imieliński, and A. Swami, ‘‘Mining association rules between sets of items in large databases,’’ in Proc. ACM SIGMOD Rec.,1993, vol. 22, no. 2, pp. 207–216.
[26] P.-N. Tan, Introduction to Data Mining. New Delhi, India: Pearson Education, 2018, pp. 328–332.

[27] E. Fix and J. L. Hodges, Jr., ‘‘Discriminatory analysis-nonparametric discrimination: Small sample performance,’’ Univ. California, Berkeley, CA, USA, Tech. Rep. ADA800391, 1952. [Online]. Available:https://apps.dtic.mil/docs/citations/ADA800391

[28] T. Joachims, ‘‘Text categorization with support vector machines: Learning with many relevant features,’’ in Proc. Eur. Conf. Mach. Learn. Berlin, Germany: Springer, 1998, pp. 137–142.

[29] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, ‘‘AndroZoo: Collecting millions of Android apps for the research community,’’ in Proc. IEEE/ACM 13th Work. Conf. Mining Softw. Repositories (MSR), May 2016, pp. 468–471.

[30] R. Genuer, J.-M. Poggi, and C. Tuleau-Malot, ‘‘Variable selection using random forests,’’ Pattern Recognit. Lett., vol. 31, no. 14, pp. 2225–2236,Oct. 2010