MOBDroid: An Intelligent Malware Detection System for Improved Data Security in Mobile Cloud Computing Environments

Noah Oghenefego Ogwara
School of Engineering, Computer and Mathematical Sciences
Auckland University of Technology
Auckland, New Zealand
fego.ogwara@aut.ac.nz

Krassie Petrova
School of Engineering, Computer and Mathematical Sciences
Auckland University of Technology
Auckland, New Zealand
krassie.petrova@aut.ac.nz

Mee Loong (Bobby) Yang
School of Engineering, Computer and Mathematical Sciences
Auckland University of Technology
Auckland, New Zealand
bobby.yang@aut.ac.nz

***Abstract*—We propose an intelligent malware detection system (MOBDroid) that aims to protect the end-user's mobile device (MD) in mobile cloud computing (MCC) environment. MOBDroid utilizes the Android Operating System (OS) permission-based security system. The APK files of 28,306 benign and malicious applications (apps) collected from the AndroZoo and RmvDroid malware repositories were used in the system development process. The apps were decompiled in order to extract their manifest files and construct a dataset comprising the permissions requested by each of the apps. We identified some unique permissions that could be used to distinguish between malicious and benign apps and performed a series of experiments using a machine learning (ML) model; the model drew on the ML.net library and was implemented in C#.net. In the experiments conducted, we obtained classification accuracy of 96.89%, a detection rate of 98.65%, and false negative rate of 1.35%. The results indicate that our model compares very favorably to other models reported in the extant literature.**

Keywords—MOBDroid, mobile cloud computing, malware detection, data security, mobile devices, machine learning

I. INTRODUCTION

Under the mobile cloud computing (MCC) paradigm, different technologies such as cloud computing (CC), mobile computing (MC) and wireless networks (WN) provide integrated services to users and organizations [1]. The popularity of the MCC technology has increased over the years due to factors such as network mobility and dynamicity, mobile device (MD) independence, ubiquitous data access, and improved data communications [2][3].

The computational services offered by the MCC environment requires data and information to be stored in virtual servers rather than in the mobile device (MD). However, MD users tend to also store sensitive and confidential information locally on their devices which makes them a prime security target. The unlimited storage and high computational power offered by MCC service providers enables users to run apps that are highly resource demanding but provide enhanced user experience, MCC users, especially Android Operating System (Android OS) users, are likely to download and install mobile applications (apps) from both official and unofficial market stores [4].

Recent trends in automation using intelligent apps together with the rapid growth of CC technologies such as the Internet of Things (IoT) has also led to increased attacks on the MCC infrastructure leading to data theft and confidential information leaks [5]. Arguably, the user end of the MCC environment (i.e., MDs) are even more exposed to attacks. For example, mobile malware developers may insert malicious code into targeted apps and publish these in popular mobile market stores. Additionally, the resource constrained nature of the MD hinders the implementation of advanced protection systems that are able to mitigate the threats posed by increasingly sophisticated malware [6]. Furthermore, most apps are designed to run without user intervention hence allowing modified app behaviours and compromised data stored in the MD to go undetected [7].

In this paper, we propose an intelligent malware detection system named MOBDroid. The system uses an ensemble of machine learning (ML) classification algorithms in order to determine whether a mobile app is malicious or benign. The classification is based on the permission requests extracted from the app; the combination of permissions can be used to differentiate between benign apps and malicious apps. We conducted experiments with a dataset that was built during the course of the research. The results of the experiments were compared to results published in the relevant literature. The performance of our system showed an improvement in comparison to some existing models. The rest of the paper is organized as follows. Section II reviews some related work. Section III describes the data collection and feature selectin methods while section IV presents the experiments and the results obtained. Section V discusses the results and their implications, outlines the research limitations, and provides directions for further research.

II. RELATED WORK

A number of techniques for tackling the ever increasing malware threats to MDs have been proposed in the literature; the methods used are based on the analysis of app components and security elements, and/or app behavioral features such as

permissions, intents, application programming interface (API) calls, system calls, kernel operations, and resource usage. For example, a model using permissions as a means of identifying malicious apps was proposed in [8]. The model (SigPID) is based on the concept of 'significant permissions' (permissions that are relevant to distinguishing between malicious and benign apps). The authors identified 22 significant permissions. The overall classification accuracy achieved was 93.62%.

Alazab et al. [5] proposed a malware detection model that used app permissions and API calls. They used three different strategies for selecting relevant API calls to improve the chances of detecting a malicious app. The research was conducted on a dataset comprising 27,891 apps. Another approach for detecting malicious application using permissions combined with hardware components, restricted API calls and network addresses is described in [9]. The authors report that their system called DREBIN achieved an overall classification accuracy of about 94%. The model was evaluated with 123,453 apps and 5,560 malware samples.

Idrees et al. [10] proposed a novel system that used a combination of permissions and the Android OS messaging object 'intent' in order to identify malicious apps, applying a statistical and ML approach. The authors tested their model with 1,745 mobile apps. The model proposed in [11] also uses intents for the purpose of the classification. As reported, the model (Androdialysis) achieved better accuracy using intents as compared to using permissions. A total of 7,406 apps were used in the experiment, with an overall detection rate of 91% when using intents and 83% when using permissions.

In [12], a framework called Deep4MalDroid was proposed that could detect malicious apps based on system calls using deep learning techniques. The framework applies the concept of weighted graphs for feature extraction. A novel approach for dynamic analysis called component traversal executes the code routines in each given Android OS based app in order to extract the Linux kernel system calls of the apps running on the device. The authors used a dataset of 3,000 Android OS based apps.

Somewhat similarly, Zhou et al. [13] addressed attack detection using dynamic feature extraction and runtime system calls. The authors applied a novel ML classifier to identify an app as a malicious one. First, information about apps were collected in the form of runtime system call records of both benign and malicious apps. Labels distinguishing between malicious and benign apps were set in the model's feature vector table; these apps were used for training and testing the model. In order to detect malware in an unknown Android OS app, the runtime system calls of the unknown app are acquired and the resulting feature vector table is fed to the detection model to evaluate the app.

Qi et al. [14] proposed a network behaviour-based malware detection system for Android OS devices. The system comprises a network behaviour monitoring module, a network behaviour analysing module, and a storage module. The monitoring module collects data to create a feature vector that identifies system activity and also reflects the impact of malware on the MD behaviour. The vector contains the process ID, the start and end times of the network connection, network traffic (up/downflow), source/destination IP addresses, protocol type, source/destination port numbers. The authors evaluated their model using 1,260 malware apps.

Another model detecting malicious activity based on behaviour (MADAM) was proposed in [15]. It analyses apps at four different levels: kernel, application, user and package. The authors tested their model with 2,800 apps and achieved an overall classification accuracy of 96%.

The security model proposed in [16] detects malicious activities in Android OS MDs by conducting simultaneously both static and dynamic analyses thus providing a rapid security response with predictive capabilities. The system is centred around four primary components: the Android OS based app, the Security Server, the Google Cloud Messaging (GCM) service, and the Analysis Module. In their experiments, the authors used 241 malicious and 241 benign apps for detection based on permission data, and 91 malicious and 95 benign apps for detection based on system call data. Another system that dynamically analysed device behaviour usage was proposed by Ribeiro et al. [17]. It monitors deviations in device behavioural characteristics based on features such as total CPU usage, memory consumption, total outgoing and incoming network traffic, and battery use. ML and statistical algorithms are used to classify an app as either benign or malicious.

To summarise, Android OS devices are widely used because of their reasonable cost, high flexibility, and open application publication policies. However, these factors also attract malware developers. The covert techniques used by malware are hard to detect by traditional antivirus software that use signature-based detection methods [8]. To protect the use of its resources, Android OS implements a permission-based security system [5] and, drawing on the findings in the literature reviewed, our proposed system (MOBDroid) uses the requested permissions of Android apps to build an effective defence against the threats faced by MDs.

III. Data Collection and Feature Selection

A. Data Collection

We constructed our own mobile app dataset by collecting Android OS apps from the two repositories AndrooZoo [19] and RmvDroid [20]. In order to retrieve the apps we built a C#.net core console program that automatically downloaded the apps' Android Package (APK) files and stored them on a server for further processing. Due to the size of the APK files, the app collection stage was time consuming and required significant storage space. An average of 800 APK files were retrieved daily over a period of eight weeks (June-July 2020); 30,000+ apps were collected.

Using the online antivirus service VirusTotal (https:// virustotal.com) the APK files were scanned in order to ascertain whether an app is benign or malicious. VirusTotal deploys a number of different antivirus engines. We labelled an app 'benign' if none of the antivirus engines flagged the APK file as malicious, and 'malicious' if at least 15 VirusTotal antivirus engines flagged it as malicious. (We assumed that at least 15 'malicious' outcomes would be

needed to identify a truly malicious app as some apps may contain potentially harmful executable code but are actually benign. A total of 28,306 apps from different app markets were classified including 9,879 benign apps (from AndrooZoo) and 18,427 malicious apps (16,306 from AndrooZoo and 2,121from RmvDroid). The size of the resulting data set is comparable to the size of the samples used in the reviewed literature.

B. Feature Extraction

The APK Easy Tool (https://apk-easy-tool.en.lo4d.com /windows) was used to decompile the APK files. The custom-built automated system automatically retrieves and extracts each app's permissions. The permissions are found in the app's manifest file which contains essential information relevant to the app's functionality. A total of 291,863 Android OS-specific permissions were extracted, out of which 132 were identified as unique. This number is comparable to the number of system permissions (165) in the Android OS API level 30 (https://developer.android.com/guide/topics/ permissions/overview).

The set of unique permissions {P1, P2, …, P132} was used to create the feature dataset used as the input to the classification component of MOBDroid as follows: each app was represented by a permissions vector {App(1), App(2)…App(i)…App(132) where i = 1, 2, …, 132 and

$$App(i) = \begin{cases} 1 & if\ permission\ is\ used\ by\ the\ app \\ 0 & if\ permission\ is\ not\ used\ by\ the\ app \end{cases}$$

C. Permission Usage Analysis

We analysed the resulting data set in order to obtain a deeper understanding of how malicious apps requested permissions to get access to and control MD resources. First, we observed that out of the 132 unique permissions used by the entire set of apps, 110 unique permissions were common to both malicious and benign apps.

We also found that most benign apps used a relatively small number of permissions. Benign and malicious apps used an average of seven and thirteen permissions, respectively. However, some benign apps also used a high number of permissions (the 'top ten' benign apps and malicious apps used from 36 to 64 and from 68 to 89 permissions, respectively).

Furthermore, we observed that 25 of the permissions were used by at least 10% of the apps (Table I, third column); the remaining 107 permissions were used less frequently. (The permissions were labelled to reflect their usage frequency and the data set was updated accordingly.)

Searching for usage patterns, we applied the Android OS permission categorization and labelled each permission as either 'normal' (not likely to pose a risk to user data if granted), or 'dangerous' (likely to pose a security and privacy risk as it enables access to sensitive information at the device level, if granted). As shown, malicious apps tend to use more of the dangerous permissions compared to benign apps (Table I, last three columns).

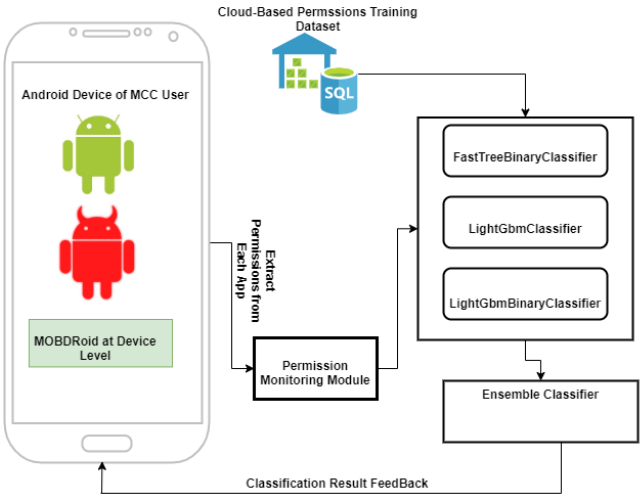

Fig. 1. MOBDroid: An Intelligent Malware Detection System.

IV. CLASSIFICATION OUTCOMES

A. Classifcation algorithms

The first five ML classification algorithms (classifiers) shown in Table II were used to build complex ML pipelines that allowed the system to evaluate apps and predict an unknown app's maliciousness. The classifiers belong to the ML.net library [18].

Classifiers C1, C2 and C4 use the decision tree algorithms and a binary classification model with two predicted possible states which is appropriate for the context (i.e., predicting whether or not an app will exhibit malicious behaviour). Classifiers C3 and C5 use a multiclassification model with two or more predicted states. Our experiments showed that all five algorithms could effectively differentiate between benign and malicious apps.

An ensemble classifier combines the results of different algorithms in order to achieve improved performance. Although the results of an ensemble classifier might not be better than that of an individual classifier in certain situations depending on the dataset, different ensembling techniques such as boosting, stacking, and bagging can be applied to obtain a better overall performance [8].

The ensemble classifier used in MOBDroid combines the results of three different ML classification algorithms in order to achieve improved performance. At the experimental stage, the stacking method was found to give the best results; hence, it was adopted in MOBDroid.

Using stacking, each individual classifier is trained independently using the same data set, and each individual predication outcome is sent to the ensemble classifier. The ensemble classifier applies the principle of majority voting to make the final prediction. As shown in Fig.1, the ensemble classifier of MOBDroid is composed of three ML classifiers (namely, FastTreeBinary, LightGBM and LightGBMBinary

TABLE I. PERMISSION USAGE

ID	Name	Overall Usage (%)	Category	Benign App Usage (%)	Malicious App Usage (%)
P1	INTERNET	99.51	Normal	98.80	99.89
P2	ACCESS_NETWORK_STATE	96.21	Normal	93.09	97.88
P3	WRITE_EXTERNAL_STORAGE	81.75	Dangerous	63.61	91.47
P4	READ_PHONE_STATE	71.85	Dangerous	25.84	96.52
P5	ACCESS_WIFI_STATE	66.69	Normal	35.59	83.36
P6	ACCESS_COARSE_LOCATION	53.11	Dangerous	24.95	68.20
P7	WAKE_LOCK	49.74	Normal	58.29	45.16
P8	ACCESS_FINE_LOCATION	48.08	Dangerous	26.72	59.53
P9	VIBRATE	45.03	Normal	34.04	50.92
P10	GET_TASKS	34.93	Dangerous	6.49	50.17
P11	RECEIVE_BOOT_COMPLETED	33.01	Normal	23.30	38.22
P12	READ_EXTERNAL_STORAGE	32.43	Dangerous	30.58	33.42
P13	CHANGE_WIFI_STATE	22.17	Normal	4.92	31.42
P14	SYSTEM_ALERT_WINDOW	21.90	Dangerous	7.78	29.47
P15	READ_LOGS	20.55	Dangerous	1.85	30.57
P16	MOUNT_UNMOUNT_FILESYSTEMS	20.43	Dangerous	1.52	30.57
P17	CAMERA	19.58	Dangerous	19.34	19.70
P18	RECORD_AUDIO	16.04	Dangerous	8.31	20.18
P19	GET_ACCOUNTS	15.98	Dangerous	19.41	14.14
P20	ACCESS_LOCATION_EXTRA_COMMANDS	15.23	Normal	1.00	22.65
P21	CALL_PHONE	14.90	Dangerous	7.62	18.81
P22	WRITE_SETTINGS	12.73	Dangerous	5.70	16.50
P23	SEND_SMS	12.03	Dangerous	2.08	17.36
P24	RESTART_PACKAGES	11.78	Normal	1.00	17.63
P25	MODIFY_AUDIO_SETTINGS	10.32	Normal	4.73	13.32

TABLE II. CLASSIFIERS USED

Classifier ID	Classifier name
C1	FastTreeBinary
C2	LightGbm
C3	LinearSymBinary
C4	LightGbmBinary
C5	StochasticDualCoordinateAscent
C6	Ensemble classifier

.

B. Experiment Setup

The feature dataset was divided into two parts: 80% of the data (22,400 apps) were used for training purposes while the remaining 20% (5,097 apps) were used for testing. The test set included 2,057 benign apps and 3,850 malicious apps. It was used as input to the five ML.net classifiers C1-C5.

The test set was uploaded to a Microsoft Azure SQL Database hosted by a cloud storage service provider. The experiments were conducted using a console based C#.net application built using version 0.5 of the ML.net library with the MS Visual Studio 2019 integrated development environment. The application retrieved the training dataset from the cloud storage in order to train the classifiers in the ML model The computer hardware used in the implementation included an Intel (R) Core (TM) i7-8700 CPU @3.20GHz, 16GB RAM, and a 500GB hard disk drive.

C. Results and Performance Evaluation

A number of experiments were run in order to identify a combination of individual classifiers that could be used in the

ensemble classifier C6. Based on these intermediate results (Table III), classifiers C1, C2 and C4 were selected. The evaluation results of the five ML.net classifiers (C1, C2, C3, C4, C5) and the proposed ensemble classifier C6 were obtained using a confusion matrix [21]. The metrics used are defined below. The following definitions were adopted:

1) True positive (TP): A malicious app is correctly identified as malicious.

2) True negative (TN): A benign app is correctly identified as benign.

3) False positive (FP): A benign app is incorrectly identified as malicious.

4) False negative (FN): A malicious app is incorrectly identified as benign.

The performance evaluation metrics were defined as follows:

1) Classification Accuracy (CA): The ratio, in percentage, of the correctly classified results to the total number of malicious and benign apps in the test set.

$$CA = \frac{TP + TN}{TP + FP + TN + FN} \times 100$$

2) Model Detection Rate (MDR): The percentage of correctly classified malicious apps in the test set containing the total number of all malicious apps.

$$MDR = \frac{TP}{TP + FN} \times 100$$

3) False Positive Rate (FPR)—Type I error: The percentage of benign apps classified as malicious in the test containing the actual number of benign apps..

$$FPR = \frac{FP}{FP + TN} \times 100$$

4) False Negative Rate (FNR)—Type II error: The percentage of malicious apps classified as benign in the test set containing all malicious apps.

$$FNR = \frac{FN}{FN + TP} \times 100$$

5) False Alarm Rate (FAR): The average of the two misclassification metrics FPR and FNR.

$$FAR = \frac{FPR + FNR}{2}$$

6) Precision Rate (PR): The ratio, in percentage of all correctly classifed malicious apps to the total number of all apps in the test set classiifed as malicious.

$$PR = \frac{TP}{TP + FP} \times 100$$

The results of the experiments that were carried out are presented in Table IV.. As shown in the table, the classification accuracy of the individual classifiers varied between 90.74% (classifier C3) and 93.36% (classifier C1).

TABLE III. CLASSIFIER PEDICTION OUTCOMES

Number of Prediction Outcomes		
Classifier	***Benign Apps (Actual: 2057)***	***Malicious Apps (Actual: 3850)***
C1	2,049	3,858
C2	2,055	3,852
C3	2,224	3,683
C4	2,031	3,876
C5	1,954	3,953
C6	2,037	3,870

Overall, the proposed ensemble classifier C6 performed better than the individual classifiers (CA=96.89%). Furthermore, the false alarm rate (FAR) results for the individual classifies varied between 9.66% (classifiers C5) and 7.35% (classifier C1). The ensemble classifier C6 showed better performance with a much lower false alarm rate (FAR = 3.88%). The ensemble classifier C6 had the best performance in the Type I and Type II metrics as well (FRP= 6.42% , FNR= 1.35%).

The model detection rate of the ensemble classifier (MDR=98.65%) compared favourably to the results of the induvial classifiers where MDR varied between 90.73% (C3), and 95.01% (C1, C5). Finally, the system's precision rate (96.64%) was higher than the precision rate of any of the individual classifiers.

V. DISCUSSION AND CONCLUSION

We compared the results obtained by the proposed system with results reported in studies that used Android OS app permissions as malware classification features. The classification accuracy of MOBDroid achieved using the test dataset was better than or comparable to the classification accuracy reported in the relevant studies reviewed (permission based systems). For example, MOBDroid's classification accuracy (96.89%) was better than the classification accuracy of MADAM [15] (96.00%), SigPID [8] (93.62%), and Androdilaysis [11] (83.00%). In particular, MOBDroid's classification accuracy was better than the classification accuracy of DREBIN [9] whose datasets are widely used for malware classification experiments. Furthermore, the FNR (type II error) achieved by MOBDroid (1.35%) was significantly lower that the type II error rates reported in the relevant work reviewed. Notably, compared to the accuracy achieved by PIndroid as reported in [10] (CA=99.8%), MOBDroid performed less satisfactorily; however, the PIndroid model uses both permissions and intents, and was tested on a very limited number of apps..

The study demonstrated that the proposed approach can be used to build an effective and relatively low overheard intelligent system capable of distinguishing between benign and malicious apps operating at the MCC user end. The system uses a single feature set and relies on static meta-analysis and it does not need 'rooting' the MD. In comparison, most existing systems require more than one feature set and device rooting.

TABLE IV. CLASSIFICATION OUTCOMEA: EVALUATION METRICS

Classifier	TP	TN	FN	FP	CA (%)	FNR (%)	FPR (%)	FAR (%)	MDR (%)	PR(%)
C1	3,658	1,857	192	200	93.36	4.99	9.72	7.35	95.01	94.82
C2	3,646	1,851	204	206	93.06	5.30	10.01	7.66	94.70	94.65
C3	3,493	1,867	357	190	90.74	9.27	9.24	9.25	90.73	94.84
C4	3,665	1,846	185	211	93.30	4.81	10.26	7.53	95.19	94.56
C5	3,658	1,762	192	295	91.76	4.99	14.34	9.66	95.01	92.54
C6	3,798	1,925	52	132	96.89	1.35	6.42	3.88	98.65	96.64

The study has several limitations. First, the number of apps initially collected was limited by the storage available. Second, only a small selection of ML.net classifiers was used. Third, the system was tested on a relatively small number of apps. Future work involving the complete development of MOBDroid for use in MDs. We aim to expand the number of classifiers considered, and the range of apps used for training and testing. Secondly, to reduce the computational overhead we intend to select features based on relevant permissions rather than using the entire list of permissions. Thirdly, to increase the classification accuracy, we will consider increasing the number of features in the dataset used in the classification process by exploiting other relevant app chacteristics as represented in the app's manifest file. This will allow us to build a hybrid intelligent malware detection system.

REFERENCES

[1] V. Moorthy, R. Venkataraman, and T.R. Rao, "Security and privacy attacks during data communication in software defined mobile clouds," Comput. Commun., vol. 153, 515-526, Mar. 2020.

[2] M. Gopichand, "Survey on sSecurity and privacy Issues in mobile cloud computing environment," Int. J. Electron. Commun. and Comput. Eng., 6(3), pp. 330-334, 2015.

[3] Z. Sanaei, S. Abolfazli, A. Gani, and R. Buyya, "Heterogeneity in mobile cloud computing: Taxonomy and open challenges," IEEE Commun. Surveys Tuts., 16(1), pp. 369-392, 2013.

[4] O. J. Nisha, and S.M.S Bhanu, "Detection of malware applications using social spider algorithm in mobile cloud computing environment," Int. J. Ad Hoc and Ubiq. Comput , 34(3), pp.154-169, 2020.

[5] M. Alazab, M. Alazab, A. Shalaginov, A. Mesleh, and A. Awajan, "Intelligent mobile malware detection using permission requests and api calls," Future Generation Comput. Syst., 107, 509-521, June 2020.

[6] N.O.Ogwara, K. Petrova, and M.L.B. Yang, "Data security frameworks for mobile cloud computing: A comprehensive review of the literature," in 2019 29th Int.. Telecommun.Netw. and Appl. Conf. (ITNAC), Auckland, New Zealand, 2019, pp. 1-4, doi: 10.1109/ITNAC46935.2019.9078007.

[7] M. B. Mollah, M. A. K. Azad ,and A. Vasilakos, "Security and privacy challenges in mobile cloud com puting: Survey and way ahead,", J. of Netw. and Comput. Appl., vol. 84, pp. 38-54, 2017.

[8] 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.,14(7), pp. 3216-3225, 2018.

[9] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C.E.R.T. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket," in 2014 Netw. & Distrib. Syst. Symp. (NDSS), San Diego, CA, vol. 14, pp. 23-26, Feb. 2014.

[10] F. Idrees, M. Rajarajan, M. Conti, T.M Chen, and Y. Rahulamathavan, " PIndroid: A novel Android malware detection system using ensemble learning methods," Comput. & Secur., 68, pp. 36-46, July 2017.

[11] A. Feizollah, N.B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of Android intent effectiveness in malware detection," Comput. & Secur., 65, pp. 121-134, Mar. 2017.

[12] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs," in 2016 IEEE /WIC/ACM Int. Conf. Web Intell. Workshops (WIW), Omaha, NE. pp. 104-111, Oct. 2016.

[13] Q. Zhou, F. Feng, Z. Shen, R. Zhou, M.Y. Hsieh, and K.C. Li, "A novel approach for mobile malware classification and detection in Android systems," Multimed. Tools and Appl., 78(3), pp. 3529-3552, 2019.

[14] Y. Qi, M. Cao, C. Zhang, and R. Wu, "A design of network behavior-based malware detection system for Android," in Sun X. et al. (eds) Int. Conf. on Algorithms and Architectures for Parallel Processing (ICA3PP 2014). Lecture Notes in Computer Science, vol. 8631, Springer, Cham, pp. 590-600, 2014. Doi: 10.1007/978-3-319-11194-0_52

[15] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "Madam: Effective and efficient behavior-based android malware detection and prevention," IEEE Trans. Dependable Secure Comput. 15(1), pp. 83-97, 2016.

[16] W.G. Hatcher, D. Maloney, and W. Yu, "Machine learning-based mobile threat monitoring and detection," in 2016 IEEE 14th Int. Conf. Software Eng. Res., Manage. and Appl. (SERA), Towson, MD, pp. 67-73, June 2016.

[17] J. Ribeiro, F.B. Saghezchi, G. Mantas, J. Rodriguez, S.J. Shepherd, and R.A. Abd-Alhameed, "An autonomous host-based intrusion detection system for Android mobile devices", Mobile Netw. and Appl., 25(1), pp. 164-172, 2020.

[18] Ahmed Z. S. Amizadeh, M. Bilenko, R. Carr, W. S. Chin, Y. Dekel, X. Dupre, V. Eksarevskiy, S. Filipi, T. Finley, and A, Goswami. "Machine learning at Microsoft with ML.NET," in ACM SIGKDD Inte. Conf. on Knowledge Discovery & Data Mining. ACM: New York, 2019, pp. 2448-2458.

[19] K. Allix, T.F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in IEEE/ACM 13th Working Conf. on Mining Software Repositories, pp. 468-471, 2016.

[20] H. Wang, J. Si, H. Li, and Y. Guo, "Rmvdroid: towards a reliable android malware dataset with app metadata," in IEEE/ACM 16th Int. Conf. on Mining Software Repositories, pp. 404-408, 2019.