International Journal on Advanced Science Engineering Information Technology

Vol.10 (2020) No. 2
ISSN: 2088-5334

DATDroid: Dynamic Analysis Technique in Android Malware Detection

Rajan Thangaveloo[a1], Wong Wan Jing[a2], Chiew Kang Leng[a3], Johari Abdullah[a4]

[a] *Faculty of Computer Science and Information Technology, University Malaysia Sarawak, Kota Samarahan, Sarawak, 94300, Malaysia*
E-mail: [1]trajan@unimas.my; [2]wanjingwng@gmail.com; [3]klchiew@unimas.my; [4]ajohari@unimas.my

***Abstract*— Android system has become a target for malware developers due to its huge market globally in recent years. The emergence of 5G in the market and limited protocols post a great challenge to the security in Android. Hence, various techniques have been taken by researchers to ensure high security in Android devices. There are three types of analysis namely static, dynamic and hybrid analysis used to detect and analyze the malicious application in Android. Due to evolving nature of the malware, it is very challenging for the existing techniques to detect and analyze it efficiently and accurately. This paper proposed a Dynamic Analysis Technique in Android Malware detection called DATDroid. The proposed technique consists of three phases, which includes feature extraction, feature selection and classification phases. A total of five features namely system call, errors and time of system call process, CPU usage, memory and network packets are extracted. During the classification 70% of the dataset was allocated for training phase and 30% for testing phase using machine learning algorithm. Our experimental results achieved an overall accuracy of 91.7% with lower false positive rates as compared to benchmarked method. DATDroid also achieved higher precision and recall rate of 93.1% and 90.0%, respectively. Hence our proposed technique has proven to be able to classify malware more accurately and reduce misclassification of malware application as benign significantly.**

***Keywords*— android malware; dynamic analysis; static analysis; hybrid analysis; malware detection.**

I. INTRODUCTION

Worldwide smartphones market is dominated by Android operating system (OS) with a staggering figure of 86.3% in 2018 as reported by IDC statistic data [1]. Android gains its popularity due to its open source concept which enables users to use it freely in Google Play as well as other third-party apps stores. This fact has encouraged millions of mobile applications developments, which have assisted in our day to day living and businesses. Owing to this huge market, attackers are interested in targeting Android platform where people spend most of their time. According to Novinson [2], Android-based malware samples has increased exponentially in 2018 compared to 2017 with a record of 5 million malwares in the first eight months alone. Goodin [3] stated in his report that hundreds of thousands of poorly secured devices with Google's Android OS had been attacked by a powerful denial-of-service attack. Besides, it is reported that there are around three hundred applications of botnet found in the official Google Play market [3]. The malwares will silently enlist the devices into the vicious network that sent junk traffic to website once the user installed the applications. The malware causes their device to become unresponsive or go offline. Moreover, providing a secure communication for 5G mobile network remain a challenge due to limited protocols to address the security issues.

It is imperative that a serious work begins intensively in Android Malware detection to defend and safeguard against these attacks. According to Skovoroda and Gamayunov [4], there are many methods based on three main techniques namely static analysis, dynamic analysis and hybrid analysis (combination of Static and Dynamic) to detect and analyze the behavior of the malicious application. The limitation of detecting Android malware with static analysis is that it does not find vulnerabilities present in the runtime environment. This is because the technique uses static signatures from the application's manifest file. As such, it is unable to accurately detect malware which does not have obvious malicious signature. In addition, most of the apps are obfuscated, or bytecode encrypted.

On the other hand, dynamic analysis is used to overcome the limitation faced by the static analysis where it performs the analysis during its runtime. Certain application will reveal their original behaviors during its run time, hence dynamic analysis helps to uncover some features that cannot be determined by static analysis. The common behaviors that concerned in the analysis is system calls, API calls, network traffic, or even file access. Among them the most popular in dynamic behaviors used is the network

traffic. In the case of network traffic analysis, Anshul and Sateesh [5] proved that HippoSMS and RogueSP-Push, capable to send and block SMS in the background silently without having to wait for runtime. Hence, this scenario posts a biggest challenge in Dynamic Analysis.

Hybrid analysis is the combination of both static analysis and Dynamic analysis technique whereby static analysis features obtained from analysis and dynamic features collected during execution of the applications will be analyzed for any malware activity in the device. Hybrid analysis has proven to yield better detection rate with high accuracy as reported by Saba et al. [6] and Mahima et al. [7] in their work respectively. Although the combination of both static and dynamic formed a better detection technique, many people prefer to use dynamic analysis as it is cost efficient in implementation compared to hybrid analysis. This is because to have a comprehensive detection on the device is impractical as it requires more hardware resources and software computation. Although some researchers analyze the relationships among feature sets such as system calls, network packets and so on, there are still weaknesses due to insufficient features for the classifier to detect the malware samples accurately.

In this paper, we proposed a dynamic analysis technique for Android malware detection called DATDroid which mainly focused on the behaviors that extracted during run time. Our proposed technique will extract the features and permissions from android application and then perform feature selection before proceeding to the classification.

The rest of the paper is organized as follows. Section II discusses the materials and method involved in our proposed framework. Section III presents the result and discussion. Finally, section IV presents the conclusion of our work.

II. Materials and Method

In this section we review three types of analysis technique used in Android malware detection. These techniques are static analysis, dynamic analysis and hybrid.

A. Static Analysis

Milosevic et al. [8] presented static analysis for Android malware detection using permission and source code analysis. The method uses two machine learning algorithms which are classification and clustering. The classification is utilized to differentiate a malware from good ware and the clustering is applied to cultivate a classification model with more data again.

Yerima et al. [9] proposed an ensemble learning Android malware detection with static analysis which is better than traditional signature-based methods for detecting unknown malware. The machine learning algorithms used a vast repository of malware samples and benign applications from a leading antivirus vendor as training data. The comparative analysis compared results of detection from several classifiers. This approach is performed without feature selection process.

Zhang and Ziao [10] introduced CSCdroid which was a contribution-level-based system call (SC) categorization. CSCdroid used the concept of contribution to quantitatively examine SCs relevance for malware identification in contrast to previous work which used all SCs to create feature vectors which determined the application's behavior. Markov chains are constructed based on the SCs to create target feature vectors. Finally, the vectors are put into the trained SVM classifier to determine the application. This work used only limited SCs.

B. Dynamic Analysis

Dynamic analysis is known as behavioral-based analysis. It is a detection that works by collecting data from system runtime during program execution. The common data collected are system call, network data, files and memory modification. Shankar et al. [11] introduced a framework named AndroTaint and worked on dynamic taint analysis to detect Android malware. AndroTaint used anomaly detection technique to identify the different components such as services, events, activities and permissions. The framework consists of two phases which are training phase and analysis phase. In training phase, taint adapter assists feature extraction and produces tags then stored them in the SysTrace log file. Later, in analysis phase automatic tagging and tainting are executed by examining the Android application with log file AppSysTrace from SysTrace log file. All behaviour and property of the taint sources are checked for any bad-ware source. If found to be bad-ware an automatic tagging process will kick start tainting. The work did not cover the basic feature like memory and CPU usage.

Martenelli et al. [12] proposed D-BRIDEMAID a lightweight application, which acts as a dynamic Intrusion Detection System (IDS), reporting the malicious application's behavior. D-BRIDEMAID is a lightweight application installed on users' android mobile phone for testing and evaluation voluntarily. After tester (user) evaluated the application, the report will be submitted. The decision of application trustworthiness is obtained by using an aggregator based on report of testers regarding every analyzed application.

Dynamic analysis is useful in analyzing the obfuscated source code of the application. Bhatia and Kausal [13] developed a syscall-capture system to analyze the behavior of the malicious application in Android by using the system call traces collected during a run time. Monkey tool is used to automate random execution of different activities on the application for a pre-specified duration of the time. While the application is running, strace utility captures all the system calls. After that, Android Monkey tool uninstalled the application after it has stopped and fetched the file. The feature set of system started to detect the behavior of the application and collect the frequency of system calls. The J48 Decision Tree and Random Forest are used on the aggregated datasets to categorize the dataset into benign or malicious applications.

Liu et al. [14] proposed an emulator-based dynamic analysis framework called RealDroid. Their framework capable to detect evasive malware behaviors and analyze Android application using automated exploration mechanism known as Android Test Engine (ATE) in large scale. Next, the framework keep tracks the system calls of target application by using process level behavior monitoring techniques for malwares and detect it.

Tangil et al. [15] proposed a framework called ALTERDROID, a dynamic analysis approach to detect the obfuscated and hidden malware components that are distributed with legitimate Android application package. ALTERDROID was based on two major differentiation methods which are fault injection and differential analysis. The original application and the automatically generated application with modification (fault) that are carefully injected are analyzed for its behavioral differences. The differential fault analysis technique is effective in detecting the stegomalware which is a malware that uses advanced hiding methods like steganography. ALTERDROID's architecture supports running various analysis task in parallel and offloading them to the cloud.

C. Hybrid Analysis

Hybrid analysis is the combination of both static analysis and dynamic analysis. Since it combines both the techniques hence its time consuming and resource constraints because it requires extended software and hardware. Martinelli et al. [12] proposed a framework analysis BRIDEMAID, which exploits both static and dynamic approaches to detect malicious applications on Android mobile devices. The static analysis uses n-grams matching to detect malicious application pattern while dynamic analysis based on multilevel monitoring include device, application and user behavior at runtime. There are three features extracted that is static, meta-data and dynamic analysis. First, the application is downloaded to do n-gram statistical analysis. It is then unpacked and undergoes batch static analysis. The malicious application is discarded during the static analysis. After that, meta-data analysis is carried out to analyze the permissions, rating, download number and developer reputation. Lastly, dynamic analysis checks the components at runtime. These components are text messages, SysCalls, installed packages, opened connections and admin authorization during the event hooking and policy enforcement. The BRIDEMAID uninstalls malicious application when it is discovered. The combination of different approaches improves the accuracy of detection.

Su et al. [16] proposed a combination of two layers (static and dynamic) analysis system. The static analysis uses WEKA tool for machine learning classification and the multiple built-in algorithms to determine the source code with optimal accuracy. At Layer 1, static analysis extracts four types of features namely permission, native-permissions, intent priority and function calls of the application from AndroidManifest.xml file. The classification process is done using WEKA tool. If at layer 1 suspicious malware behavior detected, users are notified of results. At layer 2, user uploads the suspicious malware application to the sandbox server. The dynamic analysis examines the log files in active stage for any hidden malicious action within the application and unveil it. The analysis running in the simulator will be recorded and user will be informed of the result when analysis is completed.

D. Proposed Framework

This section discusses the proposed framework for malware detection in Android mobile platform using dynamic analysis. Our proposed technique consists of three phases which are feature extraction, feature selection and classification. Figure 1 shows the proposed framework.

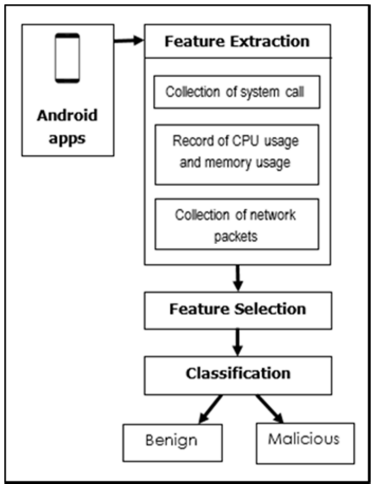

Fig 1. Proposed DATDroid framework

1) Feature Extraction: The feature extraction is the foremost necessary process as it determines the accuracy of the proposed method. The process involves extracting features from the input android application which is in APK (Android Package) file format. Our proposed technique uses dynamic analysis technique to extract the features from each application. Figure 2 illustrates the process of extraction whereby the APK files will be loaded into an emulator. This emulator is an Android virtual machine. As soon as the android application is activated, the behaviors of the application samples are logged and extracted based on the script (algorithm for feature extraction). In this work, the script contains the algorithm to extract system call, CPU usage, memory usage, and network packets. Our work used Monkey tools [17] to generate the random interaction of the applications at the runtime. Monkey is a random event creator offered as part of the toolkit of the Android developers that do not need to be modified. It is also known as a method to train the mobile applications, that sends clicks and swipes to a computer or an emulator, pseudo-random events as per Alzaylaee et al. [18]. Finally, the extracted features are logged into log files and formed a detail feature sets which will be used for the feature selection phase.

Three main features were selected out of ten features through ranker method in this proposed project to determine the characteristics of malware and benign. They are system calls, network packets, CPU and memory usage. The feature sets are extracted under each main feature for example under system call features there will be a few feature sets (sub-features) such as errors, total calls, total errors and so on. Sometimes this feature may vary based on some applications, for example benign and malware produce different call features. Thus, machine learning algorithms were used to analyze or classify their behaviors based on all these mixed contains.

The first feature that has been extracted was system call as all resources are mainly distributed through a set of system calls by the Linux kernel to the mobile applications. L. Singh and M. Hofmann [19] stated shows that by analyzing specific system call requests, the resource allocation of the

targeted application can be determined, which also differentiates the behavior of malware and benign.

Since almost most of the malicious behaviors are performed via the network interface, the DATDroid chooses this feature as a main target to study malware traits. Wang et al. [20] in their work prove that by examining the network traffic, they can uncover the exposure of delicate information by some malware applications. Furthermore, DATDroid exploits the CPU and Memory usage to detect the malware behaviors because, these two features help to understand some crucial usages by both malware and benign applications.

This paper explains the process of collecting System Calls, describes the records of CPU and memory usage, and states the process of collecting network packets.

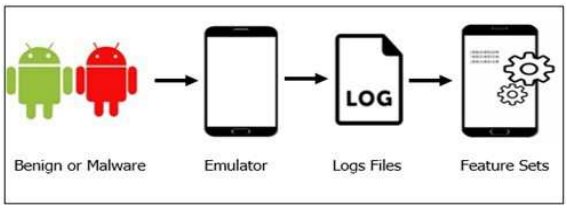

Fig 2. The flow of feature extraction

In a collection of System Call, the android x86 (emulator) is installed on the Virtual Machine (VM) and the settings are in tune for malware analysis. In order to capture the system call, the emulator must be rooted, and a utility tool called strace [21] is used. To avoid the damage of an actual android device, the malicious APKs will be installed in the emulator during the experiment. The strace utility will capture the system calls when the application is running with their multiple processes of each APK. The output of the strace consists of the system calls, the frequency of each system calls, the time taken per system call in microseconds and the percentage of the time spent by each system call during the execution of the specific application. All the strace outputs are recorded into a single Comma Separated Values (csv) file format.

The CPU and memory usage are recorded and collected during the runtime. Another utility tool called adb (Android Debug Bridge) shell [22] is used to collect the CPU usage. A list of memory usage that are captured using adb shell command from Android VM are generated. The captured memory usage details are recorded in the "proc/meminfo" directory of Linux system. The output of the memory usage file contains MemFree, Active, Inactive, Dirty, Mapped and AnonPages.

Network packet is the third feature that are essential part to be captured and recorded during the runtime of the application for feature extraction purpose. The network traffic of the packets and bytes captured are the features used to establish a malware detection metric. Tcpdump [23] is a command line which is used to capture the network packets of the specific application while it is running on the virtual machine. The captured network packets details are saved in the form of pcap file. We used Wireshark [24] as the network packet analyzer to analyze network packets and extract its details. Wireshark able to display the total transmitted packets (TX Packets), total transmitted bytes (TX Bytes), total received packets (RX Packets) and total received bytes (RX Bytes). These are the features used to determine the behavior of the malware application.

2) Feature Selection: Not all features are beneficial when combined for the classification. In addition, higher dimensionality of features requires higher processing resources. Hence, it is necessary to perform the most relevant feature selection to optimize the malware detection performance of our proposed technique. Gain Ratio Attribute Evaluator is applied in our work to perform feature selection. This method is an automated selection of attributes in the aggregated data. The Gain Ratio Attribute Evaluator is proven to work smoothly with the Random Forest classifier.

3) Classification: Coronado-De-Alba et al. [25] in their work, used one of the data mining tools called the Waikato Environment for Knowledge Analysis (WEKA) [26] for classification process. This tool provides many commonly used classifiers for android malware classification. Among many, we have selected the most popular Random Forest classifier as a classification algorithm for application behavior. S.S. Hansen [27] in their work shows that Random Forest is a collective classifier which depends on a variety of trees to decrease the classification variance and thus enhances predictive efficiency. In this classification phase, two processes namely training, and testing will be conducted. During the training process, we will train the classifier to differentiate malware from benign application and will use the trained classifier in the testing process to validate the performance.

III. Results and Discussion

This section describes the development process of our proposed DATDroid detection technique which includes software requirement, experimental setup and dataset processing. The Android application in APK format is installed in Android VM via adb command. The adb command also used to connect the Android VM to Internet Protocol (IP) address. Monkey tool [17] is scripted automatically to produce 500 random gestures and touches in 1 minute on the specified application in Android VM to depict the real action of application as if in the Android phone. Figure 3 shows the output of strace utility [21] that captured the system calls, the frequency of the system calls, time and user calls.

% time	seconds	usecs/call	calls	errors	syscall
34.78	0.056420	48	1174	114	ioctl
19.59	0.031771	24	1308		epoll_pwait
17.83	0.028923	4	8216		clock_gettime
10.68	0.017331	21	825	7	futex
6.59	0.010694	17	619		sendto
3.04	0.004931	11	429		writev
2.56	0.004147	3	1263	501	recvfrom
1.70	0.002752	2	1217		getuid32
1.28	0.002083	15	140	114	close
0.88	0.001431	4	353		write
0.62	0.001000	7	140		read
0.20	0.000324	1	276		rt_sigprocmask
0.13	0.000206	13	16		pread64
0.11	0.000171	12	14		munmap
0.02	0.000032	0	150		gettimeofday
0.00	0.000000	0	14		dup
0.00	0.000000	0	6		mprotect
0.00	0.000000	0	15		mmap2
0.00	0.000000	0	38		madvise
0.00	0.000000	0	7		fcntl64
0.00	0.000000	0	13		epoll_ctl
100.00	0.162216		16233	736	total

Fig 3. Sample of output by using strace utility

The output of the system calls was saved in a log file. The CPU usage is extracted by using 'top' command. Memory features are extracted using cat/proc/meminfo command line where it produces the detail information of the memory usage of an application. The Windows Subsystem for Linux (WSL) is used to capture the network packets of the running application in Android VM through tcpdump. The results are saved in packet capture (.pcap) file format. Then the captured network packets are analyzed in Wireshark and TCP is selected for transmission and received packets and bytes as shown in Figure 4.

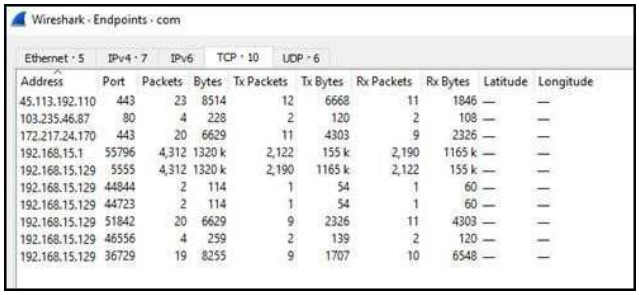

Fig 4. Wireshark analysis of captured network packets

A total of 200 dataset samples were used in this work which includes 100 benign and 100 malicious Android applications. The benign applications were downloaded from APKPure [28] market while the other 100 malicious applications were downloaded from the Android Malware Genome Project by Zhou and Jiang [29]. The benign applications are validated using VirusTotal [30] to ensure the application is clean and free from malware. Weka Tool is used for the feature selection and classification phase. We used Gain Ratio Attribute Evaluator in our work as the feature selection technique which measures the gain ratio of a corresponding class and assesses the value of an attribute. Search Method Ranker ranks the attributes of individual assessment. Thus, the order of ranking indicates the significance of each feature in the decision of correct class label as per Onik et al. [31]. The top ten attributes are selected from all the feature sets based on the highest-ranking generated by feature selection. Whereas the Random Forest classifier is used in the classification process. The classifier is used to train and test the dataset during the classification process. The collected datasets are divided into 70 percent of training sets (70 malware and 70 benign samples) and 30 percent of testing sets (30 malware and 30 benign samples). After the training phase has been completed, the trained model will be used in the testing phase to evaluate our proposed technique.

Data collected from six experiments conducted are discussed in this section. Experiment 1,3 and 5 were based on the BK method were reconducted to check their performance with three types of different features sets as mentioned in Table I. Furthermore, experiments 2, 4 and 6 were conducted based on DATDroid proposed method with different types of features combination since more than one feature has been combined and used in the proposed method. Then, the classification result is generated through Weka Tool to measure the performance matrix of our proposed technique. Table I illustrates the comparison of performance measurement between our proposed technique and existing method known as BK method by Bhatia and Kaushal [13]. Experiment 4 in Table I, proves that the highest rate was achieved with an accuracy rate of 91.7%.

The overall results of accuracy, precision, recall, and error rate are revealed in Table I. Our proposed method can achieve a higher accuracy rate than the BK method because we combined the extracted features to obtain a better result. Each experiment in our work was setup with different types of feature datasets and the details are illustrated in Table II.

TABLE I
COMPARISON OF PERFORMANCE MEASUREMENT

Android malware Detection	Experi - ments	Accuracy (%)	Precision (%)	Recall (%)	Error rate (%)
BK methods	1	78.3	90.4	63.3	21.7
	3	81.7	88.0	73.3	18.3
	5	78.3	90.5	63.3	21.7
Proposed methods	2	85.0	83.9	86.7	15.0
	4	91.7	93.1	90.0	8.3
	6	90.0	83.3	100.0	10.0

TABLE II
FEATURE SETS OF DIFFERENT EXPERIMENT

Experiment	Features	No. of features
1	System call	72
2	System call + errors + time + CPU usage + memory+ network packet	91
3	System call (feature selection)	26
4	System call + errors + time+ CPU usage + memory+ network packet (feature selection)	43
5	System call (top ten ranked attributes of feature selection)	10
6	System call + errors + time + CPU usage + memory+ network packet (top ten ranker attributes of feature selection)	10

Based on the results in Table I, we can conclude that the values of each performance measurement are varied because each experiment was conducted in different specification. For example, Experiment 1 classification with one feature set and without the feature selection process while Experiment 4 was conducted with all the feature sets and undergo feature selection as well. The result from Experiment 4 shows the highest accuracy in general and this indicates that our proposed technique achieved better performance than the existing method (BK method) with less error rate. This is due to more relevant and common features that are significant to the Android malware detection were included in our proposed experiments. Consequently, it enhances the classification process with more efficient detection and thus produces a better result.

IV. CONCLUSION

This paper presents an Android malware detection technique using a dynamic analysis called DATDroid.

Validated through a series of experiments, DATDroid is proven to perform better in terms of detection rates as compared to the former method. A total of 6 experiments were conducted with various specifications and feature selection processes. Experiment 4 is proven to be the best combination of features selected, which produced the best overall results among the six experiments. The overall accuracy of our proposed method has improved from 78.3 percent to 91.7 percent when compared to the BK method, which is without a feature selection process. In our proposed method, all the feature sets were applied with the feature selection process before the final stage of classification.

Further research can be focused on exploring more features which are not proposed in this work. Advanced features like Hypertext Transfer Protocol, Domain Name System, Transmission Control Protocol/Internet Protocol, and other memory usages pattern of each application should be extracted to improve the performance measurements of malware detection. Finally, future work should be conducted using the hybrid analysis method to get better accuracy results because the dynamic analysis can reveal the run time information while the static analysis can reveal other information that cannot be extracted through the dynamic run time process.

ACKNOWLEDGMENT

The authors would like to thank Universiti Malaysia Sarawak, MALAYSIA for funding this work through Special Short-Term Grant No. F08/SpSTG/1366/16/8. The authors would also like to thank APKPure and Android Malware Genome Project for the benign and malware samples respectively made available through their websites as well as VirusTotal for validation facilities in their website.

REFERENCES

[1] Smartphone Market Data (2019) on IDC website. [Online]. Available: https://www.idc.com/promo/smartphone-market-share/

[2] M. Novinson. (2019) The 10 Biggest Android Security Threats in 2018 on The Channel Company website. [Online]. Available: https://www.crn.com/slide-shows/security/the-10-biggest-android-security-threats-in-2018/

[3] D. Goodin. (2019) One of 1st-known Android DDos malware infects phones in 100 countries on ARC Technica website. [Online]. Available: https://arstechnica.com/information-technology/2017/08/first-knownandroid-ddos-malware-infects-phones-in-100-countries/

[4] A. Skovoroda and D. Gamayunov, "Securing mobile devices: Malware mitigation methods," *Journal of Wireless Mobile Networks, Ubiquitious Computing, and Dependable Applications.*, vol. 6, no. 2, pp. 78-97, 2015.

[5] A. Anshul and K.P. Sateesh, "NTPDroid: A Hybrid Android Malware Detector using Network Traffic and System Permissions," In *IEEE BigDataSE-18*, pp. 808-2813, 2018.

[6] A. Saba, A.S. Munam, W. Abdul, M. Amjad and S. Houbing, "SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System," *IEEE Access*, vol. 6, pp. 4321-4337, 2018.

[7] C. Mahima and K. Brij, "HAAMD: Hybrid Analysis for Android Malware Detection," In *International Conference on Computer Communication and Informatics (ICCCI-2018)* Coimbatore, India. Jan 04-06, 2018.

[8] N. Milosevic, A. Dehghantanha, and K. K. R. Choo, "Machine learning aided Android malware classification," *Computers & Electrical Engineering*, 2017.

[9] S. Y. Yerima, S. Sezer, and I. Muttik, "High accuracy android malware detection using ensemble learning," *IET Information Security*, 9(6), pp.313-320, 2015.

[10] S. Zhang, and X. Xiao, "CSCdroid: Accurately Detect Android Malware via Contribution-Level-Based System Call Categorization," In *Trustcom/BigDataSE/ICESS, 2017 IEEE*, pp. 193-200, August 2017.

[11] V. G. Shankar, G. Somani, M. S. Gaur, V. Laxmi and M Conti, "AndroTaint: An Efficient Android Malware Detection Framework using Dynamic Taint Analysis," in *ISEA Asia Security and Privacy (ISEASP)*, Jan. 2017.

[12] F. Martinelli, F. Mercaldo, A. Saracino, and C. A. Visaggio, "I find your behavior disturbing: Static and dynamic app behavioral analysis for detection of android malware," In *Privacy, Security and Trust (PST), 14th Annual Conference IEEE*, pp. 129-136, Dec. 2016.

[13] T. Bhatia, and R. Kaushal, "Malware detection in android based on dynamic analysis," In *International Conference on Cyber Security and Protection of Digital Services, IEEE* pp. 1-6. June 2017.

[14] L. Liu, Y. Gu, Q. Li and P. Su, "RealDroid: Large-Scale Evasive Malware Detection on "Real Devices," In *26th International Conference on Computer Communication and Networks (ICCCN), IEEE*, 2017.

[15] G. S. Tangil, J. E. Tapiador, F. Lombardi and R. D. Pietro, "ALTERDROID: Differentisl Fault Analysis of Obfuccated Smartphone Malware," In *IEEE Transaction on Mobile Computing*, vol. 15, no. 4, pp. 789-802, April 2016.

[16] M. Y. Su, K. T. Fung, Y. H. Huang, M. Z. Kang, and Y. H. Chung, "Detection of Android malware: Combined with static analysis and dynamic analysis," In *International Conference on High Performance Computing & Simulation (HPCS), IEEE*, pp.1013-1018, July 2016.

[17] Monkey tool (2019) on Developer Android homepage. [Online]. Available: https://developer.android.com/studio/test /monkey.html/

[18] Alzaylaee, M. K., Yerima, S. Y., & Sezer, S., "Improving Dynamic Analysis of Android Apps Using Hybrid Test Input Generation," In *International Conference on Cyber Security and Protection of Digital Services*, pp. 1-8, 2017

[19] L. Singh and M. Hofmann, "Dynamic Behavior Analysis of Android Application of Malware Detection," In International Conference on Intelligent Communication and Computational Techniques (ICCT). IEEE, 2017.

[20] S. Wang, Z. Chen, L. Zhang, Q. Yan, B. Yang, L. Peng, and Z. Jia, "TrafficAV: An effective and explainable detection of mobile malware behavior using network traffic," In IEEE/ACM 24th International Symposium on Quality of Service (IWQoS), 2016.

[21] (2019) Strace Utility website. [Online]. Available: https://strace.io/

[22] ADB shell (2019) on Developer Android homepage. [Online]. Available: https://developer.android.com/studio/command-line/adb/

[23] (2019) Tcpdump website. [Online] Available: https://www.tcpdump.org/

[24] (2019) Wireshark website. [Online]. Available: https://www.wireshark.org/

[25] LCoronado-De-Alba, L. D., Mota, R. A., & Ambrosio, P. J., "Feature Selection and Ensemble of Classifiers for android malware detection," In 8th IEEE Latin-American Conference on Communications (LATINCOM). IEEE, 2016.

[26] WEKA Tools (2019) on The University of Waikato homepage. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/

[27] S.S. Hansen, T.M. Larsen, M. Stevanovic, and J.M. Pedersen, "An approach for detection and family classification of malware based on behavioral analysis," In International Conference on Computing, Networking and Communications (ICNC). IEEE. 2016.

[28] (2019) APKPure website. [Online] Available: https://apkpure.com/

[29] Y. Zhou and X. Jiang, "Dissecting Android Malware: Characterization and Evolution," *IEEE Symposium on Security and Privacy*, 2012.

[30] (2019) VirusTotal website. [Online] Available: https://www.virustotal.com/

[31] A. R. Onik, N. F. Haq and L. Alam, "An Analytical Comparison on Filter Feature Extraction method in Data Mining using J48 Classifier" In *International Journal of Information and Education Technology*, vol. 124, no. 13, 2017.