DETECTING MALICIOUS APPLICATION USING BEHAVIOUR ANALYSIS OF MOBILE SENSORS

Sukhdev Mathur, Akshi Kumar

Abstract: Smartphones have become an inseparable part of every individual globally and the users have become increasingly dependent on these multi-functional gadgets that help in our day-to-day activities. But a user never knows what is going on inside his phone. He cannot decipher seeing a mobile application, whether it has any malicious behaviour by its appearance for any downloaded application from play store, or any third-party store. That app may be transmitting your data to a remote server without your knowledge. Even Google play store sometimes cannot detect these applications due to code obfuscation techniques. This research analyses mobile sensors' behaviour in malicious and benign mode and tries to detect if any application performs any malicious activity. Sherlock dataset has been used for the behavioural analysis by applying four supervised machine learning techniques to detect unusual behaviour and comparison has been made. We have taken two feature sets, one containing only application features, and others containing global features along with application features. We have used the F1 score as a deciding parameter for the best performance. XGBoost performs best with an F1 score of 98.82% and 98.86% on applications dataset and global dataset, respectively.

Index Terms: behavioural analysis, benign, bytes transmitted, CPU usage, malicious, memory usage, XGBoost.

1 INTRODUCTION

In 2020, there are 4.8 billion users, 3.5 billion users being smartphones owners [1]. Smartphones provide a way of communication and a prime location for the store and organizing information. As the number of applications increases for users' convenience, the number of malicious applications harming users or breaching their privacy has also risen significantly. There are 2.96 million apps on google play store, and there are some other sources available for downloading the android applications. Many applications have acquired a place in our daily routine like alarm applications, messengers, email clients, gaming applications, etc. A user never knew how an application operates in the background. It is challenging for a user to detect any malicious activity performed by an application; the application can send personal information of the user to a remote server without user consent. In late 2012, mobile malware was found on google play store named Android Dropdialer [3], this malware has self-updating capabilities. Applications infected with this malware on google play store managed to bypass their security named Bouncer. Initially, applications seem benign, had no malicious feature, and malicious components were separately downloaded from the internet known as remote payload technique. In 2017, another malware managed to bypass google bouncer [4] known as Bankbot. It can hide using code obfuscation time delay techniques. Bankbot has been removed multiples times from the Play store, but it comes back again many times even with updated capabilities. This application was designed to steal user credentials from android devices, it is capable of bypassing two-factor authentication because it can monitor text messages. Smartphone users believe that applications on google play stores are safe, but their belief makes them more vulnerable to social engineering. Generally, any malicious application like Android Dropdialor and Bankbot can be installed on a device using a remote payload technique. These applications cannot be detected using standard static techniques because the original package of application does not contain any malicious component. Using dynamic code loading to transform a non-malicious application into a malicious application makes static analysis irrelevant. By using a time-delayed or filtered deployment of the malicious payload, dynamic analysis techniques get collapsed. In this research, we aim to detect malicious applications that steal user information or spy on users. We have presented a method for detecting malicious applications using the behavioural analysis of applications i.e., how they are using the mobile resources and sensors. For conducting this research, we have used a rich dataset called Sherlock Dataset [11] provided by Ben-Gurion University. Dataset collected by two agents called Sherlock and Moriarty.

Our approach is based on monitoring applications that run on a device and detecting the sensors behaviour. Any application that runs on a device consumes resources and operate on some sensors. For anomaly behaviour detection, we have proceeded the work in two steps. First, we analyse the behaviour pattern of these Moriarty applications in benign and malicious mode. In second, we use different models and train them on the dataset and find the best model who can predict the behaviour of application most accurately. We have shown the comparison of the classifier that we have used. We have used the F1 score to select the best performing model. In the next section, the related work has been discussed, succeeded with Data set, in section 4 the methodology of the work has been elaborated followed by Analysis, Implementation and at last the Result is followed by the Conclusion.org.

2 RELATED WORK

Malicious application detection techniques have been an active area of work from past few years. Shabtai et al., in 2012, developed a framework named Andromaly; this framework has provided 88 features divided into 14 categories: touch screen, keyboard, and schedular. CPU load, messaging, power, memory, applications, calls, processes, network, hardware, binder, and led [13]. Four self-developed malicious applications and 40 benign applications were used for data collection. They performed 4 experiments using various classifiers to train their model; Bayesian Network, J48, Histogram, K-means, Logistic Regression, and Naïve Bayes.

- *Sukhdev Mathur is currently pursuing masters degree program in software engineering in Delhi Technological University, India, E-mail: devmathur1993@mail.com*
- *Akshi Kumar received the Ph.D. degree from the Faculty of Technology at the University of Delhi, Delhi, India, in 2011. She is currently an Assistant Professor with the Department of CSE, Delhi Technological University, Delhi, India. Her research interests include affective computing, sentiment analysis, social media analytics and explainable AI. E-mail: akshikumar@dce.ac.in*

For the first experiment, the entire set of benign and malicious applications were included in the ratio of 80% and 20% for training and testing. The experiment results in TPR of 99% and FPR of 0%. For the second experiment, the Training set was having 75% malicious, 75% benign applications, and the remaining set was used for testing, which gives 91% of TPR and 11% of FPR. From both the experiments, the Naïve Bayes classifier comes out as the best performer. In the third experiment, all the benign and malicious applications included in the training set lead to 91.3% of TPR and 14.7% of FPR. In the fourth experiment, 75% of malicious applications used for training, and for testing, the remaining set was used from one device, 82.5% of TPR, and 17.8% of FPR. Andromaly is capable of detecting malicious applications based on dynamic features using machine learning. Alam and Vuong, in 2013, have used behavioural features of an application on dataset collected with an android emulator that contains features from 408 benign applications and 1330 malicious applications from google play store and database of malware genome projects, respectively [14]. CPU, memory, and Binder features were used only. Network and battery features were the same in the entire dataset, so these features were excluded. Naïve Bayes, Bayesian Network, Random Forest, Multilayer Perceptron, Logistic Regression, Decision Stump and J48 classifiers were used for research. To train and test the model, authors have used a 5-fold cross-validation method. The Random forest classifier has performed the best which leads to root MSE of 0.0183%, accuracy of 99.9857%, and false positive were 2% only. Ham and Choi, in 2013, have evaluated multiple machine learning classifiers on features, network, CPU, memory, and SMS [15]. The author has used thirty benign and five malicious applications: Hostile Downloader, Spyware, a Root[1,] Spyware, and two Trojan Spywares, but the source of these applications is not mentioned. Both benign and malicious applications were run and monitored in a real environment. The feature set contains features related to Virtual memory, memory, SMS, and CPU usage. For evaluation, Naïve Bayesian, Logistic Regression, Random Forest, and SVM were used. For different malware families, Random Forest shows the best performance with a TPR of above 98.8% and FPR below 1%.Attar et al., in 2014, have used features related to battery, CPU, memory, ICMP requests, and amount of connection requests for anomaly-based detection [16]. Data collected 12 smartphones have popular applications as benign and three malicious applications. For the detection model, Gaussian Mixture Model with a Cluster-Based Local Outlier Factor was used, which results in a TPR of 100% and FPR of 0%.Milosevic et al., in 2016, have used memory usage and CPU usage features for the dynamic detection of malicious applications. Dataset used consists of 1220 malicious applications and 952 benign applications from the Google Play store and Malware Genome Project dataset, respectively [17]. The emulator had run with monkeyrunner application for data collection. The raw dataset consisted of 57 features, and after data cleaning, a clean dataset contains only seven features. Logistic regression was used with the use of a sliding window technique. For training, authors had used 571 benign applications and 727 malicious applications. The 275 benign and 304 malware applications used for testing. Finally, authors have used a validation set of 94 benign and 89 unseen malicious applications—the experiment results in TPR of 95.7% and FPR of 25%. The detection model achieved TPR of 85.5% and FPR of 17.2% with the highest F-score. Ferrante et al., in 2016 have used features used related to CPU, system calls, and memory usage. A set of benign and malicious applications has been made where malicious applications are from the Drebin dataset, and benign applications are taken from the Google Play store [18]. Dataset was collected by running the set of applications in an Android emulator with monkeyrunner application. First, the authors proceed with the K-means cluster algorithm to cluster applications based on the similarity of CPU usage features and memory usage features. After that, authors have applied Random Forest classification on applications cluster classified according to their system calls. To train their model, authors have used a set of 1000 benign applications and malicious applications. The best result has produced by 7-means clustering and Random Forest classifier of 50 trees with FPR of 28% and an accuracy of 67%.Canfora et al., in, 2016, authors have used Memory usage, CPU usage, Network usage, and Storage related features for their study [19]. A set of applications is prepared with benign applications and malicious applications from the Google Play store and Drebin data. Android emulator with monkeyrunner is used to run these applications for data collection. Authors have used Random Forest Classifiers with different types of parameters. Moreover, the authors amended the identification system using only global features, all features, and only application features. The classifier uses a global feature was comes out as the best performer. This model was trained with ten-fold cross-validations—the model has performed with 99.52% accuracy and an FPR of 0.74%.Massarelli et al., in 2017, authors have implemented a detection method with SVM based on CPU, memory, and network usage [20]. The features had system-wide and application-specific monitoring. It is unknown how many malicious applications were there in the experiment, but they had used the Drebin dataset. For data collection, an android emulator was used with monkeyrunner application to mimic the real user environment. The classifier used was a C-SVM with a radial basis function kernel. An accuracy of 82% obtained with this research but FPR is not mentioned, and the precision of the model ranges from 10% to 90% depending on the family of malware. From these researches, it can be concluded that using hardware features detection of malicious applications could be an effective way. Most research papers have used emulators and monkeyrunner application to mimic user behaviour. It has been seen that random forest classifier are the most effective technique.

3 DATASET

The data set [21] used our research is provided by the Cybersecurity Center of Ben-Gurion University. For data collection, Samsung S5 smartphones are given to 50 participants and ask them to use these devices as their primary device. These smartphones were installed with self-written malicious applications called Moriarty; they can alternate their behaviour in malicious and benign. The sherlock dataset collection starts from January 2015 and collected till December 2017, the dataset for a year is divided into four quarters, namely Q1, Q2, Q3, and Q4. This data set contains usage data of different mobile sensors like CPU usage, battery usage, location-related data, memory storage, etc. Participants were instructed to use malicious applications at least ones in a day for a few minutes. The features which are monitored are called sensors, and these sensors are group together so that at the same time, they can be sampled together. These groups

are named probes; these probes are triggered in a fixed time interval. Sensors are divided into two categories PUSH and PULL, where PUSH sensors are event-based. PULL sensors collect data periodically. We have used data of the year 2016 because, for the 2015 and 2017 dataset, Moriarty files have lesser data points. On the integration of the Moriarty probe with T4 and application probe produces insufficient dataset. We have used data of third and fourth quarters of 2016 because for quarter first and second battery features and some memory-related features are missing. Dataset is divided into two sets first contains only applications i.e., applications statistical data points and another contains both T4 i.e., systems data and application data points called global features. Both data sets merged with the Moriarty probe, which consists of self-written malicious application data points describing the session type, action type under the Moriarty application is currently running, and behavior details. Session type and action-type alternate between benign and malicious modes which we have used to label our dataset.

4 METHODOLOGY

We have removed distortions from the data set to clean the data set. After cleaning, we merged the T4 probe with applications probe based on UUID and Userid. This combined dataset merged with Moriarty probe based on nearest UUID and Userid to form global feature set, and for applications feature set Applications probes is merged with Moriarty probe. In the global feature set, we have 50 features related to memory, network, CPU, and battery features. In applications feature set, there are 27 features related to memory, network, and CPU but battery features are not present. K-nearest neighbor, logistic regression, XGBoost classifiers, and ensemble methods are used to detect malicious applications using both the data sets. Data manipulation is performed on amazon EMR service with cluster configurations of 1 master node and six slave nodes. Each node has a four-core processor, 8GBs of RAM. Pyspark is used for data processing, R for data analysis.

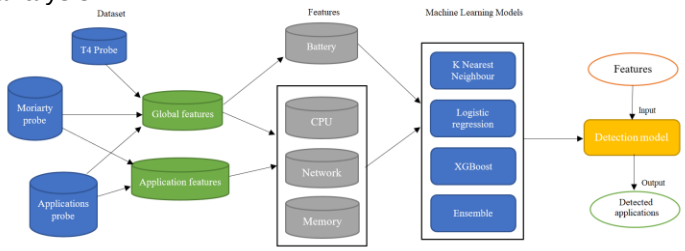

***Fig 1**. Method Flow*

5 ANALYSIS

Analysis of CPU usage, network usage, and memory usage by both malicious and benign applications, and their comparison is shown with the association plot.

5.1 CPU Usage

CPU usage is categorized in low, medium, and high. Figure 2 shows that CPU usage either low or high for malicious applications, and from benign applications, CPU usage is medium.

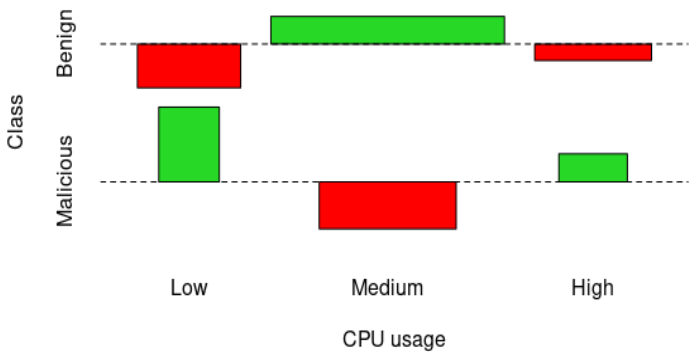

***Fig 2**. CPU usage analysis by benign and malicious applications*

5.2 Network Usage

Network usage is analyzed for bytes transmitted and bytes received by benign and malicious applications. Figure 3 shows that bytes transmitted by malicious applications are high, and bytes transmitted by benign applications are low.

Figure 4 shows that received bytes for malicious applications are lower than received bytes from benign applications.

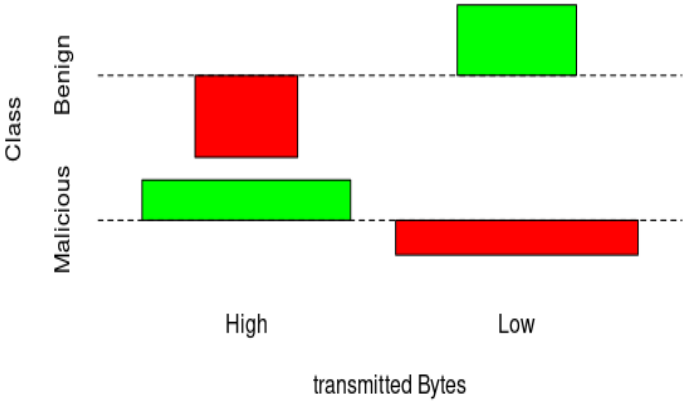

***Fig 3.** Bytes transmitted over the network by malicious and benign applications*

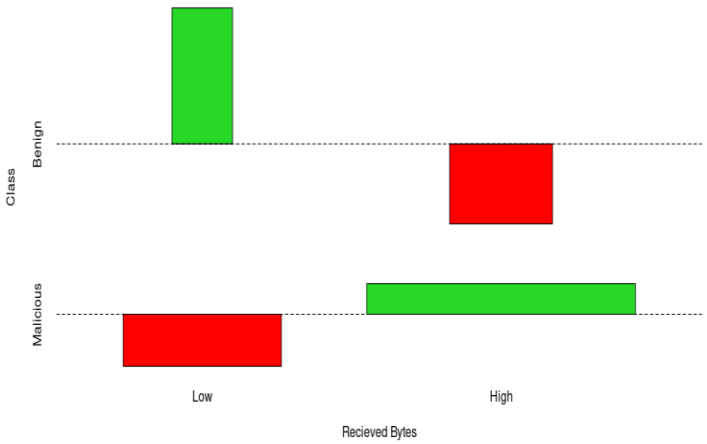

***Fig 4.** Bytes received over the network by malicious and benign applications*

5.3 Memory Usage

Memory usage is also categorized in low and high; figure 5

shows that memory used by malicious applications are lower than the memory used by the benign applications. There is a difference in the usage patterns of both benign and malicious applications from the analysis of CPU, network, and memory usage. This usage pattern has used to train the classifiers for the detection of malicious applications.

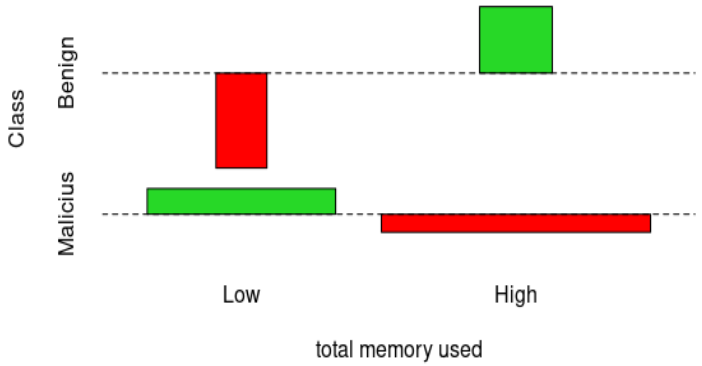

***Fig 5**. Memory used by benign and malicious applications*

6 IMPLEMENTATIONS

First, for the global dataset applications probe, T4 probe and Moriarty probe have merged, for applications dataset, applications probe and Moriarty probe have merged. Datapoints of these datasets are labeled using the session type and action type attributes. For training and testing, CPU, network, memory, and battery usage have taken in the global feature set. In applications feature set, there is CPU, network, and memory usage. For the training and testing of all four models, both feature sets have divided into the ratio of 80:20, respectively.

7 RESULTS

F1 score is used for the selection of the best model. F1-Score is the harmonic mean of precision and recall. The accuracy of the models is also measured. A comparison of all models based on the Area under Curve (AUC) value is also shown to show which model is most capable of distinguishing between benign and malicious applications.

$$\text{F1-Score} = 2 * \frac{Precision.Recall}{Precision+Recall} \quad (1)$$

$$\text{Precision} = \frac{True\ Positive}{True\ Positive+False\ Positive} \quad (2)$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive+False\ Negative} \quad (3)$$

$$\text{Accuracy} = \frac{True\ positive+True\ negative}{True\ Positive\ +True\ Negative+False\ Positive+False\ Negative} \quad (4)$$

TABLE 1
Results of classifiers with applications features set

Parameters	K-nearest neighbor	Logistic regression	XGBoost	Ensemble
Accuracy	97%	84%	98%	96%
F1 Score	98.58%	90.62%	98.82%	97.78%
Recall	9818%	84.26%	98.43%	96.20%
Precision	98.97%	98.01%	99.20%	99.42%
AUC	95.89%	90.02%	99.54%	98.42%

Table 1 shows the result of the application features set. Here XGBoost has the highest accuracy for 98%, but the F1 score is 98.82% means XGBoost has performed the best for the detection of malicious application. AUC is 99.54% given by XGBoost means XGBoost is correctly classifying 99.54% of applications, and 0.46% is incorrectly classified. Table 2 shows the result for global features set, for XGBoost accuracy is 98%, and the F1 score is 98.86%, which is highest among the four classifiers i.e., XGBoost is performing best. Moreover, AUC for XGBoost is the highest means XGBoost is correctly classifying 99.46% of applications, and 0.54% of applications are misclassified.

TABLE 2
Results of classifiers with global features set

Parameters	K-nearest neighbor	Logistic regression	XGBoost	Ensemble
Accuracy	94%	89%	98%	97%
F1 Score	96.65%	94%	98.86%	98.07%
Recall	95.90%	90.38%	98.63%	96.76%
Precision	97.41%	97.92%	99.10%	99.42%
AUC	87.27%	92.18%	99.46%	98.97%

From both the tables, it is clear that XGBoost is performing best. In the related work section, the random forest is highly efficient. In many papers, accuracy is higher than 99%, but the dataset used is collected with an android emulator, which is not a real environment data. However, in our study, Data set has taken from real users and real plate-form. Here XGboost, which is a tree-based classifier, has performed most efficiently. Global features and application features have been used for classification of malicious applications. It has been found that F1-score and AUC are almost equal for both the features set, but the Global feature set is more promising with an F1 score of 98.86%.

8 CONCLUSION AND FUTURE WORK

It has been found that CPU usage can be high or low for malicious applications. Network usage is high with malicious applications and low with benign applications. Memory usage is low with malicious applications and standard with benign applications from the analysis section. For a binary classification problem like our classification and prediction of malicious and benign applications. XGBoost, which is a tree-based classifier has performed best and produces the best result. F1-score is high for global features with a very minute difference of 0.20%, and for application features set AUC is a little high with a difference of 0.08%, so our conclusion is that global features are more reliable for the detection of malicious applications. There are some limitations as the data is collected with Samsung S5, which makes us unaware of the fact of how our model will work if data is collected with other devices. For data collection, self-written malicious applications are used. These applications are developed for research purposes only, so it is not known how the model performs with real malware. Based on behavior, we have drawn the results. In real scenarios, a benign application may use sensors like malicious applications, which leads to an incorrect result. In future, we aim to improve our model to overcome the mentioned limitations. We will research with real malware applications to improve efficiency and reliability. We will also try to collect data from multiple devices of different models so

that a huge dataset from different plate-forms can be collected to analyze the real-world scenarios and model will be deployed on real devices to test the model under real conditions.

REFERENCES

[1] "Total smartphone users in the world "https://www.bankmycell.com/blog/how-many-phones-are-in-the-world.

[2] "Total number of apps on google play store" https://www.-statista.com/statistics/266210/number-of-available-applications-in-the-google-playstore/#:~:text=The%20number%20of%20available%20apps,under%20the%20name%20Android%20Market.

[3] "Drop dialer" https://www.androidauthority.com/dropdialer-premum-rate-sms-malware-android-100783/

[4] "Google bouncer" https://www.theverge.com/2012/2/2/2766674/google-unveils-bouncer-service-to-automatically-detect-android-market

[5] "bankbot bypass google play stores bouncer" https://www.blackhat.com/docs/eu-17/webcast/10052017-scaling-security-operations.pdf

[6] "Threats to android/ types of malwares" https://kaspersky.co.in/resource-center/threats/mobile

[7] "Machine learning algorithms" https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

[8] "Performance metrics" https://www.svds.com/the-basics-of-classifier-evaluation-part-1/

[9] "Association plot" http://guianaplants.stir.ac.uk/seminar/resources/R_in_a_Nutshell_Second_Edition.pdf

[10] K. Patel and B. Buddadev, "Detection and mitigation of android malware through hybrid approach," in Security in Computing and Communications, vol. 536 of Communications in Computer and Information Science, pp. 455–463, Springer, Basel, Switzerland, 2015.

[11] Y. Mirsky, A. Shabtai, L. Rokach, B. Shapira, and Y. Elovici, "Sherlock vs moriarty: A smartphone dataset for cybersecurity research", in Proceedings of the 2016 ACM workshop on Artificial intelligence and security, ACM, 2016, pp. 1–12.

[12] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y.Weiss, ""andromaly": A behavioral malware detection framework for android devices", Journal of Intelligent Information Systems, vol. 38, no. 1, pp. 161–190, 2012.

[13] M. S. Alam and S. T. Vuong, "Random Forest Classification for Detecting Android Malware", in 2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing, Aug. 2013, pp. 663–669. DOI: 10.1109/GreenCom-iThings-CPSCom.2013.122.

[14] H.-S. Ham and M.-J. Choi, "Analysis of android malware detection performance using machine learning classifiers", in ICT Convergence (ICTC), 2013 International Conference on, IEEE, 2013, pp. 490–495.

[15] A. E. Attar, R. Khatoun, and M. Lemercier, "A Gaussian mixture model for dynamic detection of abnormal behavior in smartphone applications", in 2014 Global Information Infrastructure and Networking Symposium (GIIS), Sep. 2014, pp. 1–6. DOI: 10.1109/GIIS.2014.6934278.

[16] J. Milosevic, A. Ferrante, and M. Malek, "Malaware: Effective and efficient run-time mobile malware detector", in Dependable, Autonomic and Secure Computing, 14th Intl Conf on Pervasive Intelligence and Computing, 2nd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), 2016 IEEE 14th Intl C, IEEE, 2016, pp. 270–277.

[17] A. Ferrante, E. Medvet, F. Mercaldo, J. Milosevic, and C. A. Visaggio, "Spotting the malicious moment: Characterizing malware behavior using dynamic features", in Availability, Reliability and Security (ARES), 2016 11th International Conference on, IEEE, 2016, pp. 372–381.

[18] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Acquiring and analyzing app metrics for effective mobile malware detection", in Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics, ACM, 2016, pp. 50–57.

[19] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, and R. Baldoni, "Android malware family classification based on resource consumption over time", arXiv preprint arXiv:1709.00875, 2017.