Android Malware Detection Using Incremental Learning Approach

Shawon Kumar Saha[1],Md. Rafidul Islam Sarker[1], Md. Shohrab Hossain[1] and Husnu S. Narman[2]
[1]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Bangladesh
[2]Department of Computer Sciences and Electrical Engineering, Marshall University, Huntington, WV, USA
Email: shawonkumarsaha71@gmail.com, rafidulsarker@gmail.com, mshohrabhossain@cse.buet.ac.bd, narman@marshall.edu

***Abstract*—Android is the most popular Mobile operating system covering above 70% of all smartphone users. Due to its popularity, an increasing number of malware applications are targeting the Android platform, stealing sensitive information, accessing banking apps, etc. These malware applications can be detected with the help of machine learning algorithms by extracting features from those applications. However, previous attempts use traditional methods which require high memory overhead and computational power which is not suitable for continuous real-time data inflow. In this paper, we have proposed an incremental decision tree approach to detect mobile malware. Our model has achieved an accuracy of 93.33%, precision of 91.27%, recall of 95.83%, and F1-Score of 93.50%. Our proposed model can help reduce model retraining time when new malware samples appear. It also requires less storage for historical data while retraining the model.**

***Keywords*—Malware, Android, Static analysis, Incremental learning**

I. Introduction

Android platform is a Linux-based operating system that dominates the global market with about 3 billion users worldwide. Around 70% of all smartphone users are Android users [1]. To keep up with the demands of such a huge user base, Android applications are being made rapidly in the mobile ecosystem. Among these applications, there are many malware applications mixed within. These applications contain codes that are written intentionally to harm others. According to statistics of G DATA Mobile Security, more than 2.5 million new malware applications were identified in 2021 which is steadily increasing [2]. As we progress towards the digital age and depend on digital services such as e-commerce, banking, and medical services, we utilize many of these services on mobile phones. Average usage statistics of these digital services through smartphones are shown in Fig. 1. During the usage of these digital services, we often leave behind much personal information. These malware applications steal personal information such as contact info and banking info. They also use hijacked phones for illegal activities such as DDoS botnet, crypto mining, and so forth. Thus, finding these malware applications as soon as possible is essential for our safety.

Koli [4] proposed a supervised machine learning classification method considering request permissions, vulnerable API calls, and the presence of key information features. Almahmoud et al. [5] proposed a recurrent neural network-based approach for extracted features. Wenjia et al. [6]

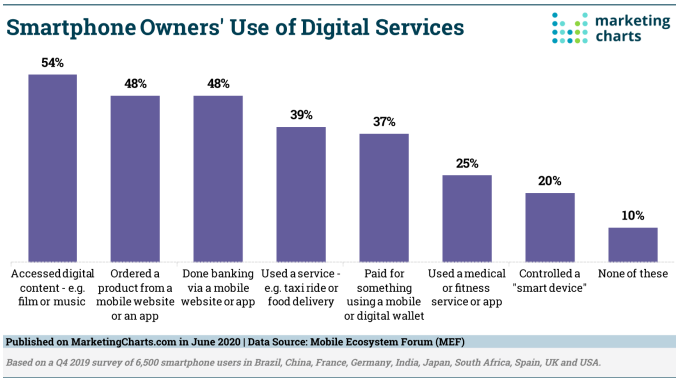

Fig. 1: Mobile usage in digital services [3].

proposed an SVM-based classification approach based on similarity scores between benign and malware applications. However, the existing works [4]–[6] have some common issues. Most of the works considered only initially extracted dataset and did not perform actions to solve the unbalanced class distribution of the dataset. These studies focused only on a small number of features and did not take into account other types of features. The main approach of these studies focuses only on offline machine learning algorithms which require the complete dataset to be present to retrain the models each time new data arrives. So the constraints of storage and computation are apparent.

The main contributions of this paper are as follows:

- We have used SMOTE algorithm to balance out the class distribution and selected the top hundred features that can be considered using information gain methods.
- We have implemented an online learning algorithm using the incremental decision tree that generates a small dataset based on the trained decision tree which can be used to retrain the decision tree along with new data.

We first attempted to mitigate the class distribution problem to create a balanced dataset using SMOTE algorithm. Then, we selected the top static features for classification. We implemented the incremental decision tree model and compared it with other well-known machine learning algorithms named Light Gradient Boosting Machine (LGBM), Extreme Gradient Boosting (XGBoost), and Decision Tree (DT) for performance comparison. Applying our incremental

decision tree, we achieved an Accuracy of 93.33%, Precision of 91.27%, Recall of 95.83%, and F1-score of 93.50%.

Our paper focuses on the premise of the combination of different features and the usability of online learning algorithms for practical scenarios where continuously new malware samples are appearing. Our paper attempts to draw the attention of researchers to the less-explored topic of online learning in malware detection.

The rest of the paper is organized as follows. In Section II, we have discussed the most related studies in this domain. In Section III, the proposed approach is explained along with dataset description, data preparation, and data preprocessing. We have presented our experimental setup and result analysis in Section IV. Section V contained the conclusion and future work.

II. Background Study

A. Terminology

From the network security perspective, Android applications are divided into two categories named Benign and Malware. Malware is an application that intentionally harms other applications or users by gaining unauthorized access to the system and causing disruption to a computer, server, client, or computer network to leak private information to others [7] [8]. Sometimes, these malware applications cause crucial damage to individuals and online-based businesses or other institutions.

These malware applications can be detected by malware analysis techniques. Generally, malware analysis in an Android environment can be classified into two major categories named static and dynamic analysis. The static analysis approach, usually considered a lightweight method for malware classification, basically focuses on file information and disassembly codes from a malware sample without sample execution. On the other hand, dynamic analysis approaches extract behavioral data from applications by executing the sample in a virtual environment and classifying based on this extracted logged behavior data [9].

B. Literature Review

The correctness of malware detection depends on feature selection and applied classification algorithms. There is much research available for malware detection in the Android environment.

Koli [4] proposed a supervised machine learning-based classification method on the extracted features from 120 benign and 150 malicious Android applications considering three extracted features named Requested permissions, Vulnerable API calls, and the Presence of key information. Machine learning algorithms like Support Vector Machine (SVM), Decision Tree, Random Forest (RF), and Naive Bayes (NB) are used for the classification of the applications. Almahmoud et al. [5] proposed a Recurrent Neural Network (RNN) based approach considering four extracted features named Permissions, API calls, system events, and permission rate from a dataset containing both malicious and benign applications. They compare their model with traditional machine learning approaches like Support Vector Machine, K-Nearest Neighbor, Naive Bayes, Random Forest, and Decision Tree for showing their model's better sides. Thangavelooa et al. [10] proposed an approach named Datdroid for the classification of Android applications considering five extracted features named system call, errors and time of system call process, CPU usage, memory, and network packets. Surendran et al. [11] proposed a novel Tree Augmented Naive Bayes (TAN) based Android malware detection technique by employing the conditional dependencies among features (API calls, permissions, and system calls). Li et al. [6] proposed a malware detection method for Android applications considering an SVM-based classification approach. They calculated the similarity scores between benign and malware applications considering suspicious API calls which were used as features in the proposed approach. Roy et al. [12] proposed a framework based on four supervised machine learning-based algorithms named Support Vector Machine, Logistic Regression, Random Forest, and K-Nearest Neighbors. They considered static API-calls features named function type, components, intent filters, and permissions from collected benign and malicious applications selected from DREBIN and CICInvesAndMal2019 datasets. Fereidooni et al. [13] proposed a lightweight machine learning classification-based model to determine malicious applications from extracted static features. They used the concept of randomized decision trees (Extra Trees-Classifier) for determining the feature importance.

C. Gap Analysis

In the above-discussed works, [4], [5], [11] considered only a few extracted features, [5], [6], [12] considered only traditional machine learning based model for classification, [4] considered only a few numbers of apk files for feature extraction, and [4] considered imbalance dataset for learning the model. In this work, we want to use an efficient online machine learning approach for classification on a balanced dataset considering more types of static features extracted from collected Android applications.

III. Proposed Approach

In this section, we discuss the information about the dataset. We also describe how the static features of the dataset of applications are extracted, preprocessed, and selection of important features. Then, we discuss the basic configuration of the architecture of the proposed model here. The basic workflow of the model is shown in Fig. 2.

A. Dataset Description

In this paper, we have used the dataset provided by Arash Habibi Lashkari et al. [14] in their paper titled "Toward Developing a Systematic Approach to Generate Benchmark Android Malware Datasets and Classification". In their dataset, they collected benign applications from the Google Play app market which were published in 2015, 2016, and 2017. They

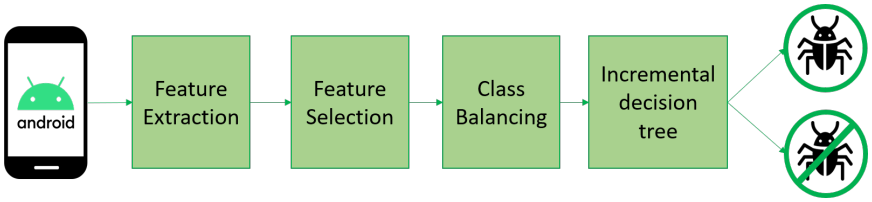

Fig. 2: Workflow diagram of the Proposed model.

collected the malware application from several sources. Their dataset consists of around 426 malware applications and 1700 benign applications. The malware applications were grouped into four major categories named Adware, Ransomware, Scareware, and SMSmalware. The dataset samples contain applications from 42 malware families belonging to these four major malware categories. Overall there were 104, 101, 112, and 109 samples from Adware, Ransomware, Scareware, and SMSmalware respectively. In our paper, we have considered 600 benign applications and all 426 malware applications.

B. Dataset Presprocessing

The preprocessing is composed of three steps: i) data extraction, ii) feature selection, and iii) data balancing. The details of the explained in the following subsections.

1) ***Data Extraction****:* The dataset contained an Android application package or apk files of benign and malware applications. These apk files needed to be extracted to analyze the source code of the apk files. To perform this activity, we used a tool called AndroPyTool [15]. AndroPyTool is a data extraction tool used for extracting static and dynamic features from the apk files. It uses different types of well-known Android analysis tools such as AndroGuard, DroidBox, Strace, FlowDroid, and so forth. We used the AndroGuard module of the AndroPyTool and extracted static features from the benign and malware files. The extracted data were then converted into a usable CSV format for further processing. After the extraction process, the class distribution looks like Table I

TABLE I: Sample and Feature number of different Apps.

App type	No of sample	No of features
Adware	119	2109
Ransomware	101	1093
Scareware	111	1955
Smsmalware	107	1577
Benign	599	4504

2) ***Feature Selection****:* The dataset is extracted from the apk files from each group together to form a complete labeled dataset. The combined dataset contained around 5238 feature columns, and these columns contained eight groups of features which are apicall, receiver, permission, activity, service, opcode, apipackage, and systemcommand. From these extracted groups of features, the top features were selected. The feature selection process was done based on the information gained value from each feature.

Information gain is a metric that calculates the reduction of entropy by transforming a dataset. The gain is defined in terms of the probability distribution of a variable in the dataset belonging to one class or another. The information gain is low for higher-probability events and high for lower-probability events.

Information gain was calculated on the features of the dataset in the context of the binary classification. Then a minimum threshold of 0.03 was used to filter the features. After filtering, from the remaining five feature groups, the top 20 features of each group were selected for analysis.

3) ***Data Balancing****:* The dataset had an imbalance in the class distribution. Because of imbalanced data, the model will learn the decision boundary ineffectively and cause the model performance to suffer. To mitigate the problem additional synthetic samples were created. The Synthetic minority oversampling technique or SMOTE is a type of data augmentation method for the minority class. This algorithm works by choosing samples that are in close proximity to the feature space. Then a line is drawn between them in the feature space. A random point is chosen between the samples along the line to create a synthetic sample. After balancing the class distribution based on binary classification, the sample distribution is shown in Table II.

TABLE II: Data sample's class distribution.

Data class	Number of samples
Benign	599
Adware	160
Ransomware	159
Scareware	139
Smsmalware	141

C. Proposed Approach

The decision tree algorithms are known for accuracy and faster train time. Rather than the computationally extensive neural network models, the decision tree-type models are often preferred for their usability. However, these tree algorithms need the whole data for building the decision tree. Therefore, in order to retrain the decision tree for new data, all past data need to be present. This causes training time to rise and wastes storage space. As shown in Fig. 3, we have followed the stated steps to mitigate the problem:

- The decision tree is first trained on available data points.

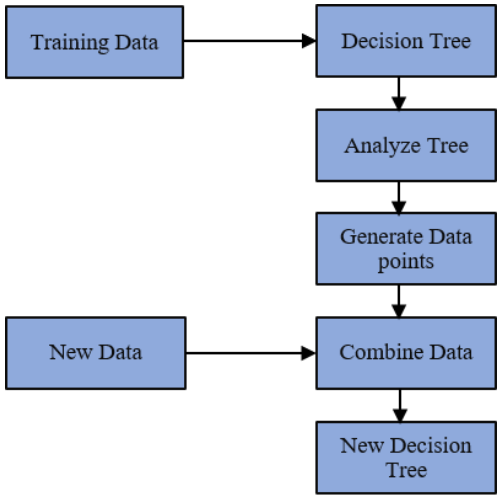

Fig. 3: Workflow diagram of the incremental decision tree.

- When new data points arrive, the structure of the tree is analyzed.
- According to the tree structure, a fixed number of data points are generated to act as the reference of the past data points.
- The new data points are combined with the generated data points to form a complete dataset.
- This newly formed dataset is used to rebuild the tree.

Thus, the tree generates a fixed number of data to act as the reference of past data points and eliminates the need for past data to be stored. By using the generated data points with new data points, the past performance of the decision tree is much affected. The model can act as a platform for streaming data and integrating the new data without much hassle.

IV. Results

A. Experimental Setup

The dataset has been split into training and testing sets to ensure that the same data used for training cannot be used again in testing. 80% data is considered training, and 20% data is considered testing for this work. This work is conducted on Google Colaboratory having free access to AMD EPYC 7B12 (RAM: 13 GB, Memory: 109 GB, L3 cache: 6384K, CPU MHz: 2249.998). The frameworks used in this study are androguard, andropytool, and Scikit learn.

B. Chosen Models for Comparison

In this study, we have compared the performance of the models of offline and incremental learning algorithms. We have considered three well-known models named Light Gradient Boosting Machine, Extreme Gradient Boosting, and Decision Tree for comparison with our proposed method. LightGBM is a gradient-boosting framework based on decision trees that increases the efficiency of the model and reduces memory usage. XGBoost is an optimized distributed-based gradient boosting library which implements machine learning algorithms under the Gradient Boosting framework [16]. A Decision Tree is a tree-structured classifier, of which internal nodes represent the features of a dataset, branches represent the decision rules, and each leaf node represents the outcome. To test and compare the performance of the models, we used the performance metrics such as Accuracy, Precision, Recall, F1-Score, TPR, TNR, FPR, and FNR.

C. Performance Metrics

Based on the prediction from the evaluated model, a confusion matrix is generated. For binary class classification, the true value is considered 1, and the false is considered 0. A Confusion matrix Shown in Table III consists of four terms: True Positive (TP), False Positive (FP), True Negative (TN), and False Negative (FN).

TABLE III: Basic Structure of General Confusion Matrix.

	PREDICTED NEGATIVE	PREDICTED POSITIVE
ACTUAL NEGATIVE	True Negative	False Positive
ACTUAL POSITIVE	False Negative	True Positive

The performance of the classification models was evaluated based on several metrics: Accuracy, Recall, Precision, F1-Score, True Positive Rate (TPR), False Negative Rate (FNR), True Negative Rate (TNR), and False Positive Rate (FPR) which are described in Eqns. (1) - (8).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$F1 - score = \frac{2 * TP}{2 * TP + FP + FN} \tag{4}$$

$$TPR = \frac{TP}{TP + FN} \tag{5}$$

$$FNR = \frac{FN}{TP + FN} \tag{6}$$

$$TNR = \frac{TN}{TN + FP} \tag{7}$$

$$FPR = \frac{FP}{TN + FP} \tag{8}$$

D. Result Analysis

In our experiment, we implemented an incremental-based decision tree. After completing the training phase, we applied the test data for prediction from the trained model. From the prediction of the model, we generated a confusion matrix shown in Fig. 4(a) and calculated other accuracy matrices from Eqns. (1) – (8) shown in Table IV.

TABLE IV: Result from Incremental Decision Tree.

Metrics	Value
Accuracy	93.33%
Precision	91.27%
Recall	95.83%
F1-Score	93.50%
True Positive Rate	95.83%
False Negative Rate	4.17%
True Negative Rate	90.83%
False Positive Rate	9.17%

*1) **Experimental Result of IDT** :* From Fig. 4(a), the trained model correctly predicts 115 positive values out of 120 positive values and correctly predicts 109 negative values out of 120 negative values. From Table IV, incremental decision tree performs better on test dataset by achieving an accuracy of 93.33%, precision of 91.27%, recall of 95.83%, F1-Score of 93.50%, TPR of 95.83% and TNR of 90.83%.

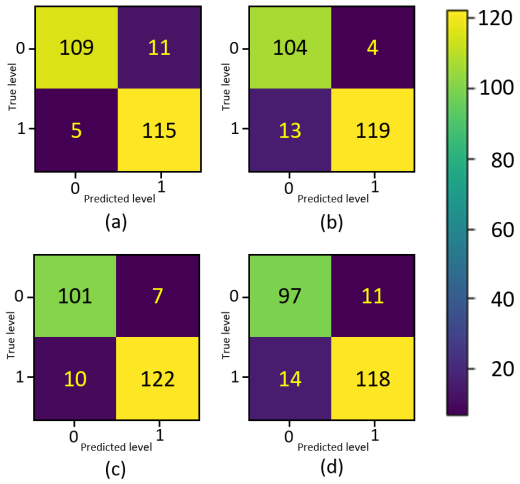

Fig. 4: Confusion Matrix of (a) Incremental Decision Tree, (b) LGBM, (c) XGBoost, (d) Decision Tree

*2) **Comparison With Other Models** :* We trained LGBM, XGBoost, and DT models on the dataset considering the same environment and parameter values used in our proposed model. Then, we applied these models to test data the same as our proposed model. The confusion matrix of LightBGM, XGBoost, and Decision Tree is shown in Fig. 4(b), 4(c) and 4(d) respectively considering False value as 0 and True value as 1. From the confusion matrix of LBGM, XGBoost, and Decision Tree, we calculated different metric values shown in Table V from Eqns. (1) – (8).

The comparison among different calculated metrics is shown in Fig. 5:

- Incremental decision tree's accuracy value (93.33%) is higher than the other three compared model's accuracy value (LGBM : 92.92%, XGBoost: 92.92% and DT: 89.58%). Considering the accuracy value, the incremental decision tree performed better.

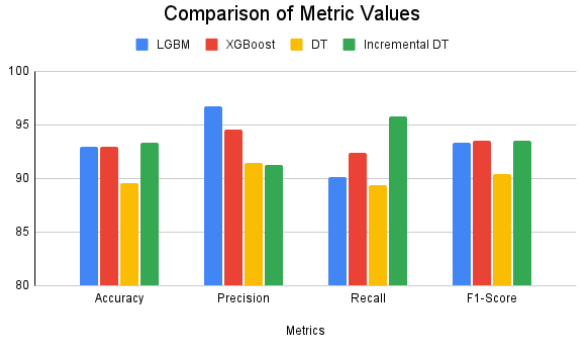

Fig. 5: Comparison of different metric values.

- Incremental decision tree's precision value (91.27%) is lower than other models' precision value (LGBM : 96.75%, XGBoost : 94.57% and DT: 91.47%). Considering precision value, the incremental decision tree did not perform well.
- Incremental decision tree's recall value (95.83%) is much higher than LGBM's (90.15%), XGBoost (92.42%) and DT (89.39%). Considering recall value, the incremental decision tree performed very well.
- Incremental decision tree's F1-Score (93.50%) is higher than LGBM (93.33%), XGBoost (93.49%) and DT (90.42%)

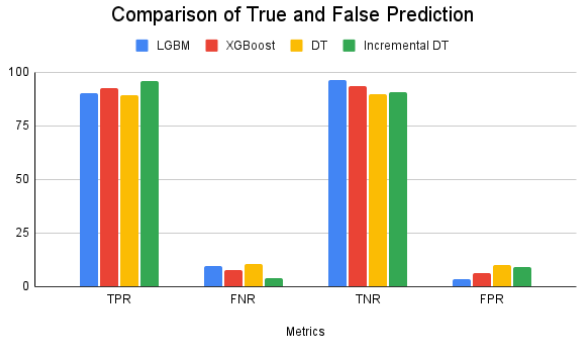

Fig. 6: Comparison of True and False prediction.

True Positive Rate (TPR) and True Negative Rate (TNR) are linearly proportional to the model's performance but False Negative Rate (FNR) and False Positive Rate (FPR) are inversely proportional to the model's performance. From Fig. 6, We can see that:

- Incremental decision tree's TPR value (95.83%) is much higher than LGBM (90.15%), XGBoost (92.42%), and DT (89.39%)'s value. Considering the TPR value, the incremental decision tree performed very well.
- Incremental decision tree's FNR (4.17%) value is much lower than LGBM (9.85%), XGBoost (7.58%), and DT (10.61%)'s value. Considering the FNR value, the incremental decision tree performed very well.

TABLE V: Result of compared models.

	LBGM	**XGBoost**	**DT**
Accuracy	92.92%	92.92%	89.58%
Precision	96.75%	94.57%	91.47%
Recall	90.15%	92.42%	89.39%
F1-Score	93.33%	93.49%	90.42%
True Positive Rate	90.15%	92.42%	89.39%
False Negative Rate	9.85%	7.58%	10.61%
True Negative Rate	96.30%	93.52%	89.81%
False Positive Rate	3.70%	6.48%	10.19%

- Incremental decision tree's TNR value (90.83%) is lower than LGBM (96.30%), XGBoost (93.52%) but higher than DT (89.81%)'s value.
- Incremental decision tree's FPR (9.17%) is higher than LGBM (3.70%), XGBoost (6.48%) but lower than DT (10.19%)'s value.

Overall our implemented model performed better on our balanced dataset than other compared traditional machine learning algorithms.

V. Conclusion

In this work, we have proposed to detect malware in the Android environment using an incremental learning approach. We have implemented an Incremental Decision Tree for better applicability in real-time data scenarios. The incremental model's computational resource cost is lower than other traditional machine learning models' computational resource cost. For understanding our model's performance over traditional machine learning models, we have compared our model with three well-known models named LGBM, XGBoost, and DT. Our Model performs better on the accuracy, recall, F1-score, TPR, and FNR values but does not perform better on precision, TNR, and FPR values.

Due to the computation capacity, we used only a small dataset in this paper. We would like to work with a much larger collection of applications in the future and use dynamic features to detect malware.

References

[1] D. Curry, "Android statistics (2022)," Aug 2022. [Online]. Available: https://www.businessofapps.com/data/android-statistics/

[2] G. Data, "G data mobile security report: More than 2.5 million new malware apps for android devices," Sep 2022. [Online]. Available: https://br.gdatasoftware.com/news/2022/02/37321-g-data-mobile-security-report-more-than-25-million-new-malware-apps-for-android-devices

[3] M. E. Forum, "Smartphone owners' use of digital services," Jun 2020. [Online]. Available: https://www.marketingcharts.com/charts/smartphone-owners-use-of-digital-services

[4] J. Koli, "Randroid: Android malware detection using random machine learning classifiers," in *2018 Technologies for Smart-City Energy Security and Power (ICSESP)*. IEEE, 2018, pp. 1–6.

[5] M. Almahmoud, D. Alzu'bi, and Q. Yaseen, "Redroiddet: android malware detection based on recurrent neural network," *Procedia Computer Science*, vol. 184, pp. 841–846, 2021.

[6] W. Li, J. Ge, and G. Dai, "Detecting malware for android platform: An svm-based approach," in *2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing*. IEEE, 2015, pp. 464–469.

[7] S. Kramer and J. C. Bradfield, "A general definition of malware," *Journal in computer virology*, vol. 6, no. 2, pp. 105–114, 2010.

[8] F. Zhong, Z. Chen, M. Xu, G. Zhang, D. Yu, and X. Cheng, "Malware-on-the-brain: Illuminating malware byte codes with images for malware classification," *IEEE Transactions on Computers*, 2022.

[9] B. Yu, Y. Fang, Q. Yang, Y. Tang, and L. Liu, "A survey of malware behavior description and analysis," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 5, pp. 583–603, 2018.

[10] R. Thangavelooa, W. W. Jinga, C. K. Lenga, and J. Abdullaha, "Datdroid: Dynamic analysis technique in android malware detection," *International Journal on Advanced Science, Engineering and Information Technology*, vol. 10, no. 2, pp. 536–541, 2020.

[11] R. Surendran, T. Thomas, and S. Emmanuel, "A tan based hybrid model for android malware detection," *Journal of Information Security and Applications*, vol. 54, p. 102483, 2020.

[12] A. Roy, D. S. Jas, G. Jaggi, and K. Sharma, "Android malware detection based on vulnerable feature aggregation," *Procedia Computer Science*, vol. 173, pp. 345–353, 2020.

[13] H. Fereidooni, M. Conti, D. Yao, and A. Sperduti, "Anastasia: Android malware detection using static analysis of applications," in *2016 8th IFIP international conference on new technologies, mobility and security (NTMS)*. IEEE, 2016, pp. 1–5.

[14] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android malware datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*, 2018, pp. 1–7.

[15] A. Martín, R. Lara-Cabrera, and D. Camacho, "A new tool for static and dynamic android malware analysis," in *Data Science and Knowledge Engineering for Sensing Decision Support: Proceedings of the 13th International FLINS Conference (FLINS 2018)*. World Scientific, 2018, pp. 509–516.

[16] B. Marais, T. Quertier, and S. Morucci, "Malware and ransomware detection models," *arXiv preprint arXiv:2207.02108*, 2022.