Computer Systems Science and Engineering

Tech Science Press

DOI: 10.32604/csse.2024.052875

ARTICLE

Modern Mobile Malware Detection Framework Using Machine Learning and Random Forest Algorithm

Mohammad Ababneh*, Ayat Al-Droos and Ammar El-Hassan

School of Computing Sciences, Princess Sumaya University for Technology, Amman, 11941, Jordan

*Corresponding Author: Mohammad Ababneh. Email: m.ababneh@psut.edu.jo

Received: 18 April 2024 Accepted: 05 July 2024 Published: 13 September 2024

ABSTRACT

With the high level of proliferation of connected mobile devices, the risk of intrusion becomes higher. Artificial Intelligence (AI) and Machine Learning (ML) algorithms started to feature in protection software and showed effective results. These algorithms are nonetheless hindered by the lack of rich datasets and compounded by the appearance of new categories of malware such that the race between attackers' malware, especially with the assistance of Artificial Intelligence tools and protection solutions makes these systems and frameworks lose effectiveness quickly. In this article, we present a framework for mobile malware detection based on a new dataset containing new categories of mobile malware. We focus on categories of malware that were not tested before by Machine Learning algorithms proven effective in malware detection. We carefully select an optimal number of features, do necessary preprocessing, and then apply Machine Learning algorithms to discover malicious code effectively. From our experiments, we have found that the Random Forest algorithm is the best-performing algorithm with such mobile malware with detection rates of around 99%. We compared our results from this work and found that they are aligned well with our previous work. We also compared our work with State-of-the-Art works of others and found that the results are very close and competitive.

KEYWORDS

Android; malware; detect; prevent; artificial intelligence; machine learning; mobile; CICMalDroid2020; CCCS-CIC-AndMal-2020

1 Introduction

It is estimated that the number of mobile phone owners globally is well over 4.8 billion, meaning that 60.42% of the world's population owns a smartphone [1]. According to Android Statistics (2024) [2], Android is the most popular operating system in the world, with over three billion active users in over 190 countries. Three-quarters of all smartphones in the world run the Android operating system. With a few exceptions, Android is the dominant platform in most countries with around 85 percent market penetration. Also, according to Kaspersky [3], in Q3 of 2023, a total of 8,346,169 mobile malware, adware, and risk-ware attacks were successfully blocked.

Copyright © 2024 The Authors. Published by Tech Science Press.
This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Android has historically been more open with fewer limitations on applications, thus contributing to its popularity. This same reason also contributed to malware abusing this openness to spread malware and conduct malicious or criminal activities.

There are multiple approaches to malware detection. The traditional approach includes signature-based, dynamic analysis, behavior anomaly, identity, and access control analysis. The other approach is by using AI and machine learning to build models and use algorithms to detect malware running automatically and autonomously on or attempting to penetrate mobiles and computers. Compared to Machine Learning solutions, traditional malware detection methods are ineffective and take more time to detect unseen malware [4].

In this article, we extend our previous work in [5], in which a machine learning (ML) model is introduced to deal with malicious code in devices running the Android Mobile Operating System and tools. The original model was built around the CICMalDroid2020 dataset. In this work, we use the newer and more comprehensive CCCS-CIC-AndMal-2020 dataset and test our methodology on the same 4 categories of malware seen in our previous work. The dataset has 200,000 instances of malware collected from various sources. To ensure its balance, another 200,000 instances were added to the dataset which includes 14 categories of malware. Each category contains several variants or families of a total of 154. Each family includes several instances, as shown in Table 1. Every malware has two instances in the dataset; the first one has feature values collected before restarting the virtual machine running the malware, which represents the case when the malware gets into the system and starts running immediately if it can. The second one has feature values collected after the virtual machine is restarted, which is the case when malware tries to avoid detection by postponing its activity when it discovers that it is running in a virtual machine or sandbox [6].

Table 1: Malware categories in CCCS-CIC-AndMal-2020

	Malware category	Number of families	Number of samples
1	Adware	48	47,210
2	Backdoor	11	1538
3	File infector	5	669
4	No category	–	2296
5	PUA	8	2051
6	Ransomware	8	6202
7	Riskware	21	97,349
8	Scareware	3	1556
9	Trojan	45	13,559
10	Trojan-banker	11	887
11	Trojan-dropper	9	2302
12	Trojan-SMS	11	3125
13	Trojan-spy	11	3540
14	Zero-day	–	13,340
	Total	127	195,624

We conducted feature analysis to select the minimal number of features with the maximum contribution to the detection decision and found that the elimination of some features has contributed to the increase in accuracy and performance. The importance of our work can be understood when implemented by current Intrusion Detection System (IDS) systems to detect new categories of malware more effectively.

The main contributions of this research can be summarized as follows:

a) Used a new dataset with new categories of malware families.
b) Generated a subset of new categories and conducted all necessary data engineering to guarantee balance.
c) Conducted feature analysis to identify the most significant features essential for the highest possible rate of detection.
d) Tried multiple Machine Learning algorithms in order to find the optimal algorithm suitable to the categories of malware under focus.

2 Related Work

The authors in [7] proposed a deep-learning framework called DE-LADY to detect malware in Android using dynamic features. The framework comprises four major modules: dynamic analysis, feature extraction and preprocessing, deep learning classifier and datasets, and evaluation metrics. The performance of seven algorithms (Linear Support Vector Machine, Naive Bayes, Decision Tree, K-Nearest Neighbor, Extreme Gradient Boosting, and Random Forest) was evaluated, and it was found that the proposed approach had a performance edge. Their approach was evaluated on 13533 APK (Android application package), with an accuracy of 98.08%.

In [8], the authors proposed a method for malware detection that makes use of the Android manifest permission analysis along with a static analyzer and APK Tool's de-compilation capability for extracting APK code at the code level. Different ML algorithms were trained using the dataset from the AndroZoo repository and were tested on 5243 samples. The proposed approach was evaluated using the Support Vector Machine (SVM), Random Forest (RF), K-means, and Naive Bayes (NB). The highest accuracy was obtained when using RF with an accuracy of 82.5% precision and 81.5% recall.

In [9], the authors proposed a hybrid model for malware detection combining Gated Recurrent Unit (GRU) and Deep Belief Network (DBN). The authors used DBN and GRU to process dynamic and static APK features, and then Back Propagation (BP) neural networks used the results of DBN and GRU as input for classification. Although the performance and accuracy of this approach were positive it is hindered by its demand for high computational power.

In [10], the authors used the CICInvesAndMal2019 dataset in their proposed approach, with 396 malicious and 1126 benign applications selected for testing. The dataset contained static features: Intents and Permissions. Five machine learning algorithms were implemented: Random Forest, Decision Tree, K-Nearest Neighbor (KNN), SVM, and NB. Experimental results showed that Random Forest achieved the highest accuracy of 96.05%.

In [11], the authors used a DREBIN dataset containing 11,120 APKs of which 5560 were malicious with 179 different malware families and 5560 were benign. For feature selection, they used a substring-based method using the Decision Tree, Gradient Boosting, Random Forest, and Extended Randomized algorithms; Random Forest achieved the best results with an accuracy of 97.24%.

The authors in [12] proposed MAPAS, a technique for the classification of malicious activities based on the behavior of malware and benign applications using Call Graphs and Convolutional Neural Networks (CNN). CNN was used to find common feature representations from the Application Programming Interface (API) call graphs, while a lightweight classifier based on the Jaccard Similarity Coefficient performed the detection. They compared their technique with MaMaDroid which is an Android malware detection approach, MAPAS obtained a higher accuracy (91.27%) than (84.99%) for the MaMaDroid method.

The authors in [13] used the Graph Neural Networks (GNN) based classifier to generate API graph embedding as a way to show the efficacy of graph-based classification. Multiple machine learning and deep learning algorithms were trained using graph embedding with "Intent" and "Permission" to detect Android malware. In the CICMaldroid and Drebin datasets, the obtained classification accuracy was 98.3% and 98.68%, respectively. The authors then proposed VGAEMalGAN, an approach based on the Generative Adversarial Network (GAN) to target GNN Android malware classifiers based on two types of networks; the Substitute Detector (SD) and the Generator (GN). The GN's goal was to produce fake data that is often malicious yet resembles genuine data and cannot be identified by SD (which plays the adversary role). Conversely, SD aims to discriminate legitimate data from fraudulent data produced by GN (in this case playing the defender role). This approach showed how Android malware detection could be strengthened against adversarial inference by retraining the model with GN samples after being classified as malware attacks.

The authors in [14] proposed a novel (Tree Augmented Naive Bayes) TAN-based hybrid model that utilizes the conditional dependencies among relevant dynamic and static features to determine whether an application is malicious. They trained three ridge regularized Logistic Regression classifiers. The results showed that the model succeeded in identifying malware with an accuracy of 97%.

The authors in [15] proposed DATDROID, a malware detection technique based on dynamic analysis using Random Forest as 3 stages: the first is to extract the features, the second is to select the features using Gain Ratio Attribute Evaluator, and the third is a classification. In the classification stage, the data was divided using the 70:30 fold for training and testing respectively and the approach obtained a classification accuracy of 91.7% with 0.9 recall and 0.931 precision.

In [16], the authors presented the CCCS-CIC-AndMal-2020 dataset and employed Entropy Analysis of Dynamic Characteristics to classify and detect malware. They used entropy-based behavioral analysis to classify the behavior of 12 renowned Android malware categories comprising 147 families using the CCCS-CIC-AndMal-2020 dataset. Their research used 6 classes of dynamic characteristics including memory, API, network, Logcat, battery, and process to classify and characterize Android malware. Their approach successfully determines the behavior of malware categories.

In [17], the authors used Deep Image Learning to detect malware using the CCCS-CIC-AndMal-2020 dataset. The authors proposed an image-based deep neural network method to classify and characterize Android malware instances from the dataset. They successfully demonstrated the effectiveness of their approach with an accuracy of 93.36% and a log loss of less than 0.20.

The authors in [18] proposed a new Subgraph Networks (SGN) based technique to detect malware using the Android Function Call Graph (FCG). For increased model robustness, a denoising mechanism 1-Lipschitz was used to filter out the attack noise. The model achieved the highest accuracy, recall, and F1-scores thus ensuring that the proposed approach will counter abnormal attacks.

In [19], the authors proposed the AMDDL model, which is a novel Convolutional Neural Network (CNN) architecture for highly accurate Android malware detection. This model uses 215 features from the Drebin dataset and obtained an accuracy score of over 99%, thus, designating AMDDL as one of the foremost models offers an impactful solution for malware protection.

The authors in [20] suggested applying machine learning to detect and classify sophisticated malware attacks targeting Android platforms. The aim was to prevent increasing attack vectors such as malware and anti-dynamic variants through the utilization of time-series KronoDroid data with features that are extracted from actual device execution and additionally using feature selection and ExtraTree classifier with Random Forest model to obtain a high accuracy for malware detection and categorization.

The authors in [21] designed GuardDroid, a system for lightweight malware detection on Android Internet of Things (IoT) devices, in which the static features extracted from the programs were analyzed. They used the Drebin dataset for training the model, they also used the Recursive Feature Elimination for feature selection. Using a Random Forest classifier they then achieved a high level of accuracy in malware classification. The work is noteworthy in terms of the efficiency of required recourses.

In [22], the authors proposed an approach that is based on feature engineering together with machine learning for the detection of Android malware. They used dex2jar and Apktool to extract the features from the apps (permissions, API calls, and intents). Then they used feature selection techniques like gain ratio and chi-squared test to determine the best features for classification. The paper evaluates the proposed framework on two benchmark datasets Drebin and TUANDROMD. The results show that both RF and SVM have achieved an accuracy level of over 98%. Table 2 summarizes the surveyed research.

Table 2: Summary of surveyed articles

Reference	Year	Dataset	Feature selection	ML/DL algorithm	Best result	Main contribution
[7]	2021	Bespoke	NA	Deep learning model with 4 hidden layers (200 neurons each)	F1-score: 98.84%, Accuracy: 98.08, Error rate: 1.92%	Introduction of De-LADY, a system using deep learning that is resilient to obfuscation methods, outperforming existing machine learning approaches.
[8]	2021	AndroZoo	NA	Random forest	Precision: 82.5%, Accuracy: 81.5%	Using android manifest file permissions, highlighting the significant improvements over commercial anti-virus tools.
[9]	2020	Applications downloaded from Google Play, APKpure, and public malware-sharing websites	NA	DBN-GRU hybrid model	Accuracy: 99.4%	Proposal of a hybrid deep learning model combining DBN and GRU for android malware detection, effectively detecting obfuscated malware and improving detection capability.

(Continued)

Table 2 (continued)

Reference	Year	Dataset	Feature selection	ML/DL algorithm	Best result	Main contribution
[10]	2021	CICInvesAndMal2019 dataset	Principal component analysis (PCA)	Random forest	Accuracy: 96.05%	A technique focusing on static features, identifying random forest as the best performer.
[11]	2018	DREBIN dataset	Substring-based method	Random forest	Accuracy: 97.24%, TPR: 96.88%, FPR: 2.39%, F1-score: 97.23%, Precision: 97.58%	A malware detection method utilizing substring-based feature selection, demonstrating random forest's superior performance on the DREBIN dataset.
[12]	2018	Some samples from VirusShare, AMD, and Google Play Store	Grad-CAM	CNN + Jaccard similarity coefficient	Accuracy: 91.27%	MAPA is a practical android malware detection system demonstrating higher accuracy in detecting unknown malware and various types of malware effectively.
[13]	2022	CICMalDroid 2020 and Drebin	selectFrom Model method	GNN	Accuracy: 98.33% (CICMaldroid), 98.68% (Drebin)	Using graph neural networks, the proposed VGAE-MalGAN algorithm achieved high accuracy in detecting malware in IoT devices.
[14]	2020	Some samples from Drebin, AMD, Github, and AndroZoo datasets	NA	Tree augmented Naive Bayes	Accuracy: 91.7% Precision: 93.1% Recall: 90%	TAN-based hybrid malware detection mechanism combining static and dynamic features, achieving high accuracy in detecting malicious applications.
[15]	2020	Applications downloaded from APKPure and the android malware genome project	Gain ratio attribute evaluator	Random forest	Accuracy: 91.7%	DATDroid is a dynamic analysis technique, that achieves good accuracy and significantly reduces misclassification.
[16]	2021	CCCS-CIC-AndMal-2020	NA	Decision tree	Precision: 98.4% F1-score: 98.3% Recall: 98.3%	EntropLyzer, an entropy-based behavioral analysis technique, achieves high precision and recall values.
[17]	2020	CCCS-CIC-AndMal-2020	ExtraTree classifier, Gini importance value, and best k features	Convolutional neural network (CNN)	93.36% accuracy, log loss less than 0.20.	DIDroid, a deep learning-based android malware classification system achieving a high accuracy.
[18]	2024	VirusShare and AndroZoo	NA	GCN	Accuracy: 97.2% Recall: 97% Precision: 97.3% F1-score: 97.1%	A novel approach to enhancing the robustness of deep learning models by mitigating the impact of adversarial attacks.
[19]	2024	Drebin	NA	CNN	Accuracy: 99.92% Recall: 99.16% Precision: 98.61% F1-score: 98.88% Loss: 0.08	AMDDLmodel is one of the models with very high accuracy to detect Android malware which is proven effective in detecting malicious applications.

(Continued)

Table 2 (continued)

Reference	Year	Dataset	Feature selection	ML/DL algorithm	Best result	Main contribution
[20]	2024	The subset of the KronoDroid dataset	ExtraTree classifier	Random Forest	98.03% accuracy (detection), 87.56% accuracy (classification)	An effective supervised machine learning model for detection and category classification.
[21]	2024	Drebin	Recursive feature elimination (RFE)	Random forest	Accuracy: 99% Recall: 98.7% Precision: 100% F1-score: 99.3%	GuardDroid: Lightweight and efficient android IoT malware detection system using machine learning.
[22]	2024	Drebin and TUAN-DROMD	Gain-ratio and Chi-square test	Random forest	Drebin: Accuracy: 98.8% Recall: 97.6% Precision: 99.3% F1-score: 94.9% TUANDROMD: Accuracy: 99% Recall: 98.7% Precision: 100% F1-score: 93.6%	An advanced feature engineering framework that improved the performance of machine learning models.

The above review shows that threats are constantly evolving. This evolution affects the effectiveness of real-time detection. Up-to-date datasets are not easy to obtain and process to offer high-reliability levels. It can also be observed that high accuracy detection rates were possible to achieve through the use of a large number of features. Complex computation powers and high execution times are required.

In our work, we introduce a new machine learning model, which takes advantage of behavioral analysis of malware APKs using a reduced set of features that improves the model's accuracy and efficiency over other frameworks. Our tight methodology, which includes data collection, pre-processing, and a comparison of different datasets and classifiers is robust and obtains positive results.

3 Research Methodology

In this work, we propose a machine learning model for mobile malware detection relying on the analysis of the malware's APK behavior. We compare various machine learning algorithms to find the most appropriate algorithm for such attacks. In our proposed approach, we construct a machine-learning model and conduct comparative analyses with other machine-learning classifiers, algorithms, datasets, and malware categories. Our proposed machine learning-based model consists of multiple phases:

3.1 Datasets and Data Collection

In this work we used the CICMalDroid2020 and the CCCS-CIC-AndMal-2020 datasets; both from the Canadian Institute for Cybersecurity (CIC) at the University of New Brunswick (UNB), Canada [23]. The first dataset was collected from December 2017 to December 2018 with 11,598 instances and 471 features; it has five categories of malware, namely: Riskware, Adware, SMS malware, Banking malware, and Benign, as in Table 3 below.

Table 3: Common malware categories in CICMalDroid2020

	Category	Description
1.	Riskware	Applications contain vulnerabilities that can be exploited by malicious actors.
2.	Adware	Applications can be installed on devices after some web browsing or other activities and start showing advertisements.
3.	Trojan-SMS	Applications that users install without being aware that those applications have malicious aspects and usually come through SMS [24].
4.	Trojan-banker	Applications try to steal credentials from financial institution clients or gain access to their financial information [25].

The second dataset is the latest dataset generated by CIC, UNB [6]. Table 1 shows the 14 categories of malware in the dataset, the number of malware families within each category and the number of instances in each category-family combination.

3.2 Data Preprocessing

The CCCS-CIC-AndMal-2020 dataset, as claimed by its developers, is balanced. For the instances of categories of malware under study in this work, the Synthetic Minority Over-sampling Technique (SMOTE) and Spread-Sub-Sample were used to balance the subset and instances, Fig. 1 shows the status before and after balancing.

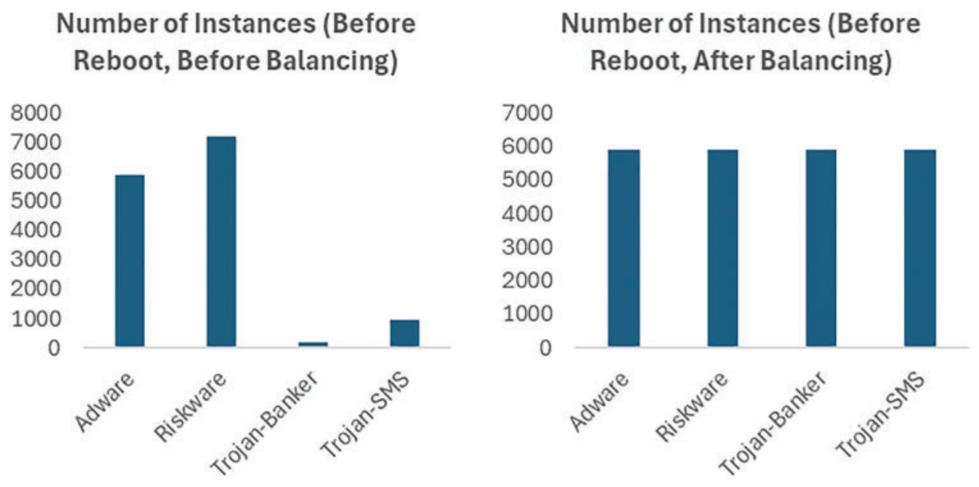

Figure 1: (Continued)

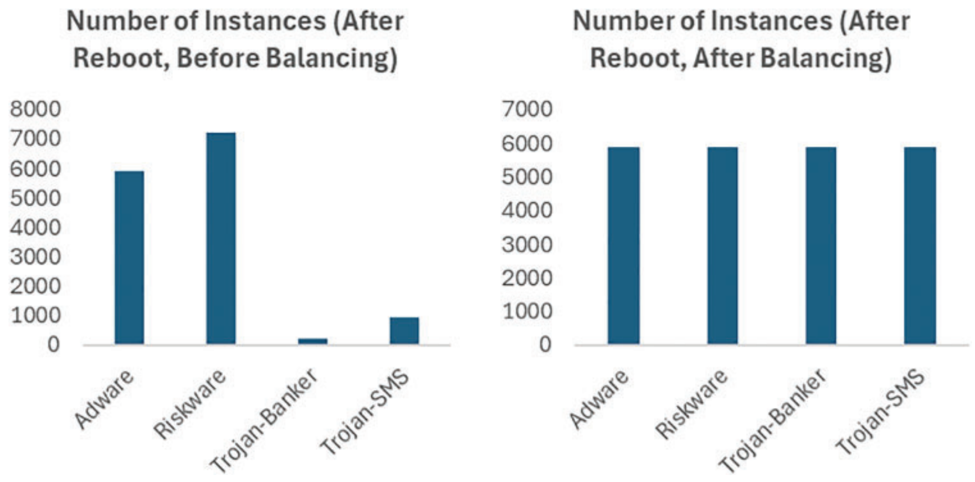

Figure 1: Handling imbalanced data

3.3 Feature Selection

The Information Gain Attribute Evaluator (GAN) and the Correlation-based Feature Selection (CFS) Subset Evaluator for feature selection were used on both datasets. GAN was used to rank the attributes, then the CFS subset evaluator was used for attribute selection. In the analysis phase, we removed 2 features. The removal of the "ResolveIntent" feature did not affect the results. However, the removal of the "shutdown" feature had a positive impact on the results as we will see in the next section. As a result, our model uses only 27 significant attributes, as shown in Table 4.

Table 4: Selected features

No.	Feature name	No.	Feature name
1.	pread64	15.	getSubscriberId
2.	gettimeofday	16.	getInstallerPackageName
3.	munmap	17.	isAdminActive
4.	sigprocmask	18.	getInTouchMode
5.	Iseek	19.	getInstalledPackages
6.	pwrite64	20.	getReceiverInfo
7.	CREATE_FOLDER	21.	FS_ACCESS(CREATE READ)
8.	getDisplayInfo	22.	FS PIPE ACCESS(WRITE)
9.	getApplicationInfo	23.	getWifiServiceMessenger
10.	statfs64	24.	queryIntentServices
11.	FS_PIPE_ACCESS	25.	setComponentEnabledSetting
12.	_newselect	26.	sendAccessibilityEvent
13.	getActivePhoneType	27.	Class
14.	flock		

3.4 Classification Algorithm Evaluation

In this stage, we used some of the most popular classifiers to detect and classify malware. The used classification algorithms and a brief description of each are as follows:

a) Random Forest (RF): in RF, the result of a new entry is voted on based on the classifiers' predictions. RF incorporates numerous tree classifiers.
b) J48 is a decision tree that generates classification or regression models in the form of a tree structure. It gradually divides a dataset into smaller and smaller groups while, at the same time, constructing a decision tree. The end result is a tree containing decision nodes and leaf nodes. A leaf node represents a classification or conclusion [26].
c) Instance-Based Learning (IBK), which is also called k-Nearest Neighbors (KNN), is a lazy algorithm. The distance is the factor that determines the similarity between two data points [27].
d) Naive Bayes (NB) is a supervised classification algorithm. It is based on the Bayes theorem and the naive assumption of conditional independence among variable pairs [28].

The model phases are illustrated in Fig. 2 below.

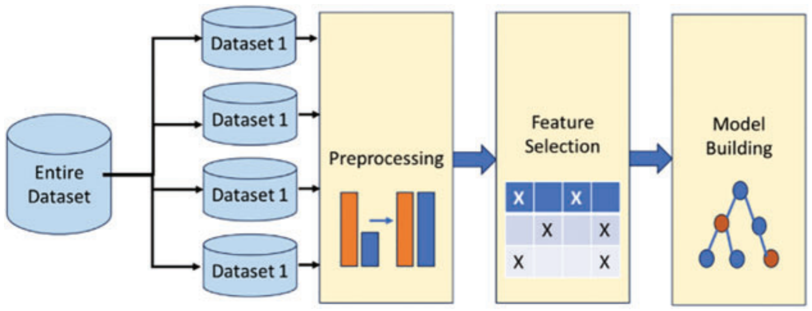

Figure 2: The machine learning approach for malware detection model

In our experimental evaluations, we used the K-fold cross-validation technique, which splits the instances into several subsets where the parameter K indicates the number of folds. A value of 10 was selected as the K parameter. The overall results were assessed using a number of metrics, including accuracy, recall, and precision.

4 Experiments and Results

In this work, we conducted experiments on dynamic features collected in the dataset. In order to monitor the behavior of malware more effectively, the dataset was divided into two categories: **before** and **after** the virtual machine reboot. This is significant because new malwares are able to detect the environment in which they are running and conceal their existence; for example, if they detect a VM or sandbox host they will stop their activity and may take other actions to conceal their presence from the VM.

4.1 Experiment on CCCS-CIC-AndMal-2020

In this experiment, we extracted malware samples of the same categories from the new CCCS-CIC-AndMal-2020 dataset that existed in the older CICMalDroid2020 dataset that was used in our previous work and applied our methodology to them to detect malware. The malware samples under focus here were related to rows: 1, 7, 10, and 12 of Table 1, which shows these categories and a brief description of them.

4.1.1 Experiment before Machine Restart

The first part of the experiment was conducted on the "Before" instances in the dataset, which represent the behavior or dynamic features of malware before restarting the virtual machine hosting the malware.

Table 5 and Fig. 3 show the results of the implementation of the five algorithms on the "Before" dataset using the previously used four categories with the 10-fold cross-validation technique. It can be seen that Random Forest achieves the highest performance values.

Table 5: Results of the implementation of the five algorithms on the "Before" dataset—test mode: 10-fold cross-validation

Algorithm	Accuracy	Recall	Precision	F-measure
Random forest	98.84%	98.80%	98.80%	98.80%
IBK	96.68%	96.70%	96.70%	96.70%
J48	95.65%	95.70%	95.70%	95.70%
Logistic regression	72.34%	72.30%	72.60%	72.30%
NB	52.75%	52.80%	58.60%	50.10%

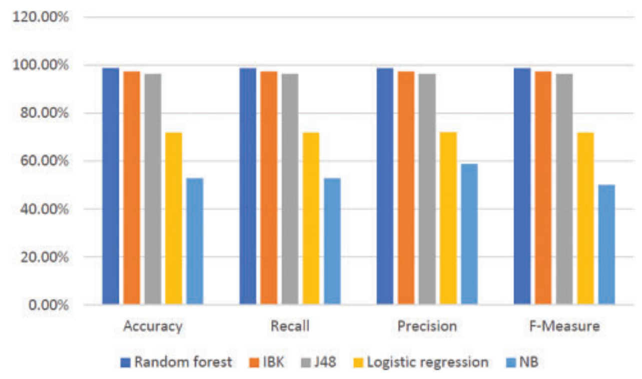

Figure 3: Results of the implementation of the five algorithms on the "Before" dataset—test mode: 10-fold cross-validation

Table 6 and Fig. 4 show the results of the implementation of the five algorithms on the "Before" dataset using the previously used four categories using the 80 training, 20 testing cross-validation technique. It can also be seen that Random Forest achieves the highest results across all metric values.

Table 6: Results of the implementation of the five algorithms on the "Before" dataset—test mode: 80 training, 20 testing

Algorithm	Accuracy	Recall	Precision	F-measure
Random forest	98.74%	98.70%	98.70%	98.70%
IBK	97.32%	97.30%	97.30%	97.30%
J48	96.34%	96.30%	96.30%	96.30%
Logistic regression	71.83%	71.80%	72.00%	71.80%
NB	52.84%	52.80%	58.7%	50.10%

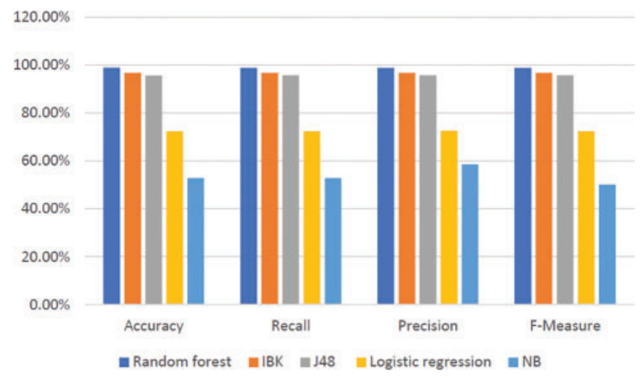

Figure 4: Results of the implementation of the five algorithms on the "Before" dataset—test mode: 80 training, 20 testing

4.1.2 Experiment on Dataset after Machine Restart

The second part of the first experiment was conducted on the "After" records in the dataset, which represent the behavior or dynamic features of malware after restarting the virtual machine hosting the malware. Table 7 and Fig. 5 show the results of the implementation of the five algorithms on the "after dataset" using the previously used four categories with the 10-fold cross-validation technique. It can be seen that Random Forest achieves the highest performance figures.

Table 7: Results of the implementation of the five algorithms on the "After" dataset—test mode: 10-fold cross-validation

Algorithm	Accuracy	Recall	Precision	F-measure
Random forest	98.93%	98.90%	98.90%	98.90%
IBK	97.81%	97.80%	97.80%	97.80%
J48	96.54%	96.50%	96.50%	96.50%
Logistic regression	72.23%	72.20%	72.40%	72.1%
NB	57.72%	57.70%	59.90%	55.80%

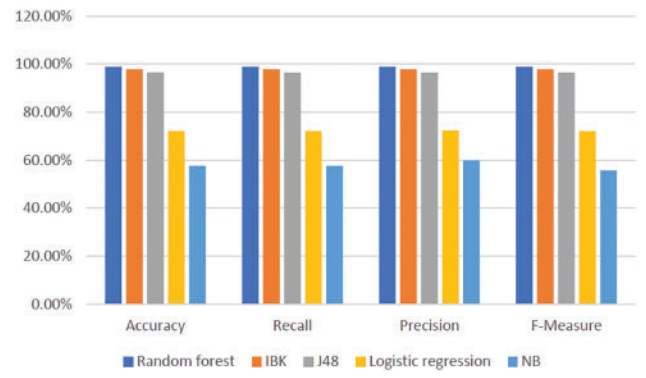

Figure 5: Results of the implementation of the five algorithms on the "After" sataset—test mode: 10-fold cross-validation

Table 8 and Fig. 6 show the results of the implementation of the five algorithms on the "after" dataset using the previously used four categories with the 80 training, 20 testing cross-validation technique. It can also be seen that the Random Forest algorithm is the one achieving the highest results across all metric values.

Table 8: Results of the implementation of the five algorithms on the "After" dataset—test mode: 80 training, 20 testing

Algorithm	Accuracy	Recall	Precision	F-measure
Random forest	98.49%	98.50%	98.5%	98.5%
IBK	96.93%	96.90%	96.90%	96.90%
J48	95.74%	95.70%	95.70%	95.70%
Logistic regression	71.65%	71.70%	71.90%	71.50%
NB	58.36%	58.40%	60.80%	56.40%

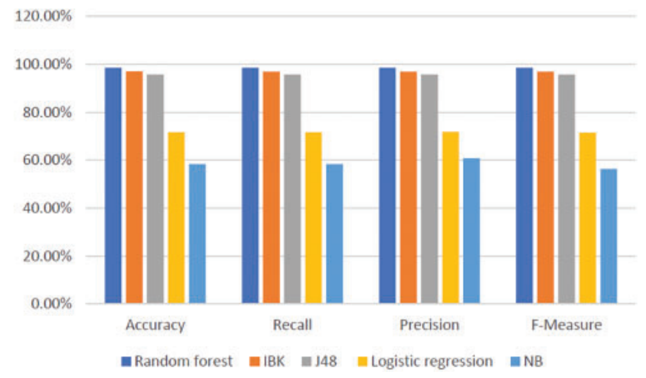

Figure 6: Results of the implementation of the five algorithms on the "After" dataset—test mode: 80 training, 20 testing

4.2 Feature Analysis

In addition to feature selection, we conducted further analysis to see if we could improve the accuracy and further reduce the time needed to classify by including or excluding features.

4.2.1 Dataset before Reboot

When we used the new dynamic-before-reboot subset from the CIC dataset, initially, the number of features was 144. It was reduced to 29 after using the CFS Subset Evaluator. After more feature analysis was performed, the accuracy increased when the following features were removed:

a) API_Database_android.content.ContextWrapper_databaseList
b) API_Database_android.database.sqlite.SQLiteDatabase_update
c) API_DeviceInfo_android.telephony.TelephonyManager_getSimSerialNumber
d) API_IPC_android.content.ContextWrapper_startActivity
e) API_DeviceData_android.os.SystemProperties_get.

We tried to remove additional features, but this resulted in a drop in accuracy. Hence, we could still achieve good results using 24 features instead of the 29 selected by the selection algorithm. Table 9 and Fig. 7 show the results after removing the 5 features with test mode: 10-fold cross-validation.

Table 9: Results with reduced features test mode: 10-fold cross-validation

Algorithm	Accuracy	Recall	Precision	F-measure
Random forest	98.75%	98.80%	98.80%	98.80%
IBK	97.06%	97.10%	97.10%	97.10%
J48	96.37%	96.40%	96.40%	96.40%
Logistic regression	68.74%	68.70%	69.30%	68.80%
NB	52.17%	52.20%	57.50%	48.90%

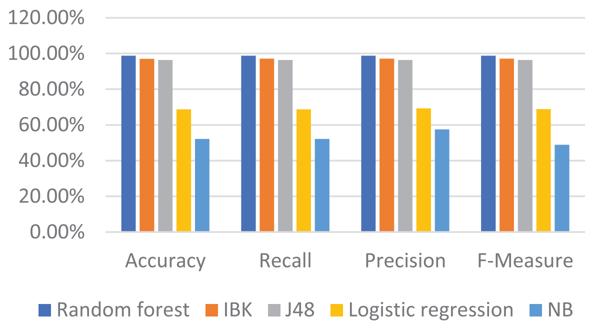

Figure 7: Results with reduced features test mode: 10-fold cross-validation

Table 10 and Fig. 8 show the results after removing the 5 features with test mode: 80 training, 20 testing.

Table 10: Results with reduced features test mode: 80 training, 20 testing

Algorithm	Accuracy	Recall	Precision	F-measure
Random forest	98.86%	98.90%	98.90%	98.90%
IBK	96.52%	96.50%	96.50%	96.50%
J48	96.06%	96.10%	96.10%	96.10%
Logistic regression	69.06%	69.10%	69.80%	69.20%
NB	52.04%	52.00%	57.40%	49.00%

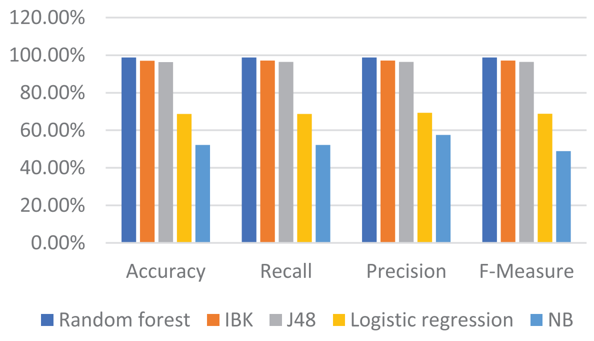

Figure 8: Results with reduced features test mode: 80 training, 20 testing

4.2.2 Dataset after Reboot

When we used the new dynamic-after-reboot subset from the CIC dataset, initially, the number of features was 144. It was reduced to 28 after using CFS. After more feature analysis was applied, the accuracy increased when the following features were removed:

a) API_DeviceInfo_android.os.Debug_isDebuggerConnected
b) API_SMS_android.telephony.SmsManager_sendTextMessage
c) API_IPC_android.content.ContextWrapper_startActivity
d) API_DeviceData_android.os.SystemProperties_get

We tried to remove more features, but the accuracy decreased. So, we could still achieve good results using 24 features instead of the 28 selected by the selection algorithm. Table 11 and Fig. 9 show the results after removing the 4 features using the 10-fold cross-validation test mode.

Table 11: Results after reboot with reduced features test mode: 10-fold cross-validation

Algorithm	Accuracy	Recall	Precision	F-measure
Random forest	98.96%	99.00%	99.00%	99.00%
IBK	97.63%	97.60%	97.60%	97.60%
J48	96.40%	96.40%	96.40%	96.40%
Logistic regression	69.10%	69.10%	69.80%	68.90%
NB	55.74%	55.70%	57.60%	53.10%

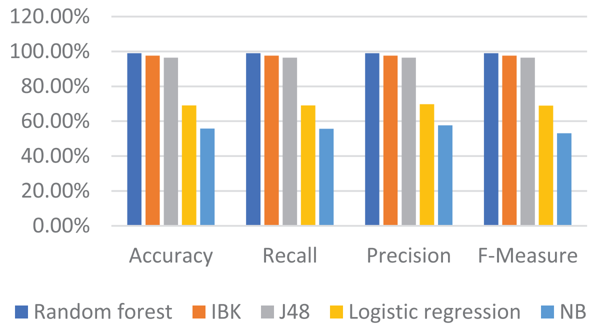

Figure 9: Results after reboot with reduced features test mode: 10-fold cross-validation

Table 12 and Fig. 10 show the results after removing the 4 features using the test mode: 80 training, 20 testing.

Table 12: Results after reboot with reduced features test mode: 80 training, 20 testing

Algorithm	Accuracy	Recall	Precision	F-measure
Random forest	98.36%	98.40%	98.40%	98.40%
IBK	96.88%	96.90%	96.90%	96.90%
J48	95.32%	95.30%	95.30%	95.30%
Logistic regression	68.71%	68.70%	69.50%	68.40%
NB	56.56%	56.60%	58.80%	53.90%

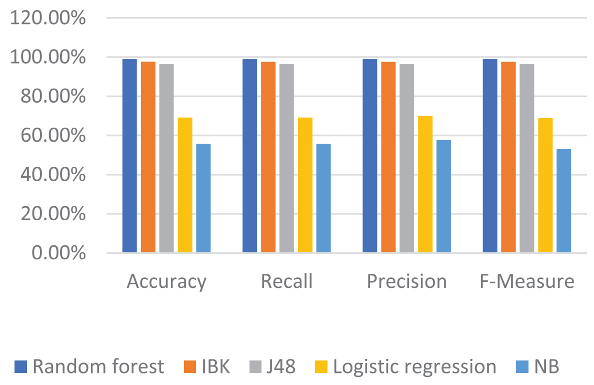

Figure 10: Results after reboot with reduced features test mode: 80 training, 20 testing

Table 13 provides a comparison of our proposed approach with other state-of-the-art methods.

Table 13: Comparison with other state-of-the-art methods

Reference	Year	Dataset	Number of features	Feature selection	ML/DL algorithm	Best result
[7]	2021	Their own dataset	NA	NA	Deep learning model with 4 hidden layers (200 neurons each)	F1-score: 98.84%, Accuracy: 98.08%, Error rate: 1.92%
[8]	2021	AndroZoo	NA	NA	Random forest	Precision: 82.5%, Accuracy: 81.5%
[9]	2020	Applications downloaded from Google Play, APKpure, and malware-sharing websites	351 features	NA	DBN-GRU hybrid model	Accuracy: 99.4%
[10]	2021	CICInvesAnd Mal2019 dataset	14 features	Principal component analysis (PCA)	Random forest	Accuracy: 96.05%
[11]	2018	DREBIN dataset	8 features	Substring-based method	Random forest	Accuracy: 97.24%, TPR: 96.88%, FPR: 2.39%, F1: 97.23%, Precision: 97.58%
[12]	2018	Some samples from VirusShare, AMD, and Google Play Store	NA	Grad-CAM	CNN + Jaccard similarity coefficient	Accuracy: 91.27%
[13]	2022	CICMalDroid 2020 Drebin	NA	selectFrom Model method	GNN	Accuracy: 98.33% Accuracy: 98.68%
[14]	2020	Some samples from Drebin, AMD, Github, and AndroZoo datasets	108 features	NA	Tree augmented Naive Bayes	Accuracy: 97%
[15]	2020	Application download from APKPure and the android malware genome project	5 features	Gain ratio attribute evaluator	Random forest	Accuracy: 91.7% Precision: 93.1% Recall: 90%
[16]	2021	CCCS-CIC-AndMal-2020	141 features	NA	Decision tree	Precision: 0.984 F1-score: 0.983 Recall 0.983
[17]	2020	CCCS-CIC-AndMal-2020	2237 features	ExtraTree classifier, Gini importance value and best k features	Convolutional neural network (CNN)	Accuracy: 93.36% Log loss of less than 0.20

(Continued)

Table 13 (continued)

Reference	Year	Dataset	Number of features	Feature selection	ML/DL algorithm	Best result
[18]	2024	VirusShare and AndroZoo	NA	NA	GCN	Accuracy: 97.2% Recall: 97% Precision: 97.3% F1-score: 97.1%
[19]	2024	Drebin	215 features	NA	CNN	Accuracy: 99.92% Recall: 99.16% Precision: 98.61% F1-score: 98.88% Loss: 0.08
[20]	2024	The subset of the KronoDroid dataset	50 features	ExtraTree classifier	Random forest	98.03% accuracy (detection), 87.56% accuracy (classification)
[21]	2024	Drebin	40 features	Recursive feature elimination	Random forest	Accuracy: 99% Recall: 98.7% Precision: 100% F1-score: 99.3%
[22]	2024	Drebin and TUAN-DROMD	54 & 61	Gain-ratio and Chi-Square test	Random forest	Accuracy: 98.8% Recall: 97.6% Precision: 99.3% F1-score: 94.9%
Our proposed framework	2024	CICMalDroid 2020 (After and before reboot) CCCS-CIC-AndMal-2020 (Before reboot) CCCS-CIC-AndMal-2020 (After reboot)	26 features 24 features 24 features	Information gain attribute evaluator and the CFS subset evaluator	Random forest	Accuracy: 98.61% Precision: 98.6% Recall: 98.61% Accuracy: 98.84% Precision: 98.8% Recall: 98.8% F1-score: 98.8% Accuracy: 98.93% Precision: 98.9% Recall: 98.9% F1-score: 98.9%

By analyzing the comparison table, it is clear that our proposed model has outperformed state-of-the-art models, although there are two models that have higher accuracy figures. Nonetheless, given the number of features used, we find that they are very large in comparison to the number of features that exist in our model (215 and 351 features), and this, in turn, may contribute to the complexity of the model which requires additional computational processing and will also require more time in extracting features from APKs and more time to classify.

5 Conclusions and Future Work

In this work, we developed a Machine Learning methodology to enhance previous malware classification by using a new dataset with new categories of Android malware families and larger samples. Machine learning classifiers were used to detect malware. Our methodology successfully classified Android APKs as malware or benign applications by using features from the CCCS-CIC-AndMal-2020 and the older CICMalDroid2020 dataset. In the model we developed, each Android application is classified using 4 machine learning classifiers; Random Forest, NB, J48, and IBK

(KNN). Accuracy, Recall, Precision and F-measure metrics were calculated through the confusion matrix (TP, TN, FP and FN) of each classification algorithm used.

In order to mitigate the risk of model-to-data bias and to improve the generalizability of the machine learning models, we experimented with the CCCS-CIC-AndMal-2020 and the older CICMalDroid2020 dataset wherein the latter, newer dataset contains a higher number of instances as well as additional malware categories to be classified.

Experimental results show that the Random Forest classifier achieved the highest results among the other algorithms in malware detection. This achievement can be explained by its use of multiple decision trees of various subsets of the given dataset then averages out the observations to enhance accuracy. The Random Forest algorithm aggregates predictions from each tree and generates a final output based on majority voting.

The performance advantage seen in the Random Forest algorithm in the malware detection task can be attributed to different factors, especially those related to dataset characteristics combined with the inherent strengths of the algorithm. Specifically, it contains numeric features that represent various software characteristics from the samples. The Random Forest's performance advantage is that it can automatically select and combine the most discriminating characteristics from the ensemble of decision trees, which enables it to handle high-dimensional numerical data easily. Furthermore, the Random Forest ensemble, which relies on the combination of the forecasts of multiple uncorrelated trees, provides higher robustness to noise and outliers, which are common for the malware data, because of the ever-evolving nature of malicious software.

Random Forest algorithm keep their positions in malware detection through the predilection to tricky relations between many features. This is of the utmost importance in the case of discriminating benign from malicious software since the difference is mostly revealed through the complex patterns of features and their interactions. Unlike Logistic Regression (LR) and NB which assume features to be independent and linearly related, the Random Forest model can deal with such complex relationships, hence giving a better performance.

In future work, a whitelist and a blacklist can be implemented to minimize the time and effort to classify instances. Also, this model can be tried on other datasets related to Android or other mobile systems.

Acknowledgement: The authors acknowledge the continuous support of their institution.

Funding Statement: The authors received no specific funding for this study.

Author Contributions: The authors confirm their contribution to the paper as follows: study conception and design: Mohammad Ababneh, Ayat Al-Droos; data collection: Mohammad Ababneh, Ayat Al-Droos; analysis and interpretation of results: Mohammad Ababneh, Ayat Al-Droos, Ammar El-Hassan; draft manuscript preparation: Mohammad Ababneh, Ammar El-Hassan. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The data that support the findings of this study are available from the corresponding author, Mohammad Ababneh, upon reasonable request.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. Turner "How many smart phones are in the world?," 2024. Accessed: Feb. 20, 2024. [Online]. Available: https://www.bankmycell.com/blog/how-many-phones-are-in-the-world

[2] D. Curry "Android statistics (2024)," 2024. Accessed: Feb. 20, 2024. [Online]. Available: https://www.businessofapps.com/data/android-statistics/

[3] A. Kivva "IT threat evolution in Q3 2023 mobile statistics," 2024. Accessed: Feb. 20, 2024. [Online]. Available: https://securelist.com/it-threat-evolution-q3-2023-mobile-statistics/111224/

[4] E. J. Alqahtani, R. Zagrouba, and A. Almuhaideb, "A survey on android malware detection techniques using machine learning algorithms," in *Sixth Int. Conf. Softw. Def. Syst. (SDS)*, Rome, Italy, Jun. 10–13, 2019, pp. 110–117.

[5] A. Droos, A. Al-Mahadeen, T. Al-Harasis, R. Al-Attar, and M. Ababneh, "Android malware detection using machine learning," in *13th Int. Conf. Inf. Commun. Syst. (ICICS)*, Irbid, Jordan, Jun. 21–23, 2022, pp. 36–41. doi: 10.1109/ICICS55353.2022.9811130.

[6] S. Mahdavifar, A. F. Abdul Kadir, R. Fatemi, D. Alhadidi, and A. A. Ghorbani, "Dynamic android malware category classification using semi-supervised deep learning," *The 18th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC)*, Calgary, AB, Canada, Aug. 17–24, pp. 515–522, 2020

[7] V. Sihag, M. Vardhan, P. Singh, G. Choudhary, and S. Son, "De-LADY: Deep learning-based android malware detection using dynamic features," *J. Internet Serv. Inf. Secur.*, vol. 11, no. 2, pp. 34–45, May 2021.

[8] N. Herron, W. B. Glisson, J. T. McDonald, and R. K. Benton, "Machine learning-based android malware detection using manifest permissions," in *Proc. 54th Hawaii Int. Conf. Syst. Sci.*, Honolulu, HI, USA, Jan. 5–8, 2021. doi: 10.24251/HICSS.2021.839.

[9] T. Lu, Y. Du, L. Ouyang, Q. Chen, and X. Wang, "Android malware detection based on a hybrid deep learning model," *Secur. Commun. Netw.*, vol. 2020, no. 6, pp. 1–11, Jan. 2020. doi: 10.1155/2020/8863617.

[10] A. Sangal and H. K. Verma, "A static feature selection-based android malware detection using machine learning techniques," in *Int. Conf. Smart Electr. Commun. (ICOSEC)*, Trichy, India, Sep. 10–12, 2020, pp. 48–51. doi: 10.1109/ICOSEC49089.2020.9215355.

[11] M. Rana, S. S. M. M. Rahman, and A. H. Sung, "Evaluation of tree based machine learning classifiers for android malware detection," in *Int. Conf. Comput. Collect. Intell.*, Bristol, UK, Sep. 5–7, 2018, pp. 377–385.

[12] J. Kim, Y. Ban, E. Ko, H. Cho, and J. H. Yi, "MAPAS: A practical deep learning-based android malware detection system," *Int. J. Inf. Secur.*, vol. 21, no. 4, pp. 725–738, Aug. 2022. doi: 10.1007/s10207-022-00579-6.

[13] R. Yumlembam, B. Issac, S. M. Jacob, and L. Yang, "IoT-based android malware detection using graph neural network with adversarial defense," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 8432–8444, May 2023. doi: 10.1109/JIOT.2022.3188583.

[14] R. Surendran, T. Thomas, and S. Emmanuel, "A TAN based hybrid model for android malware detection," *J. Inf. Secur. Appl.*, vol. 54, no. 3, pp. 102483, 2020. doi: 10.1016/j.jisa.2020.102483.

[15] R. Thangavelooa, W. W. Jinga, C. K. Lenga, and J. Abdullaha, "DATDroid: Dynamic analysis technique in android malware detection," *Int. J. Adv. Sci., Eng. Inf. Technol.*, vol. 10, no. 2, pp. 536, 2020. doi: 10.18517/ijaseit.10.2.10238.

[16] D. S. Keyes, B. Li, G. Kaur, A. H. Lashkari, F. Gagnon and F. Massicotte, "EntropLyzer: Android malware classification and characterization using entropy analysis of dynamic characteristics," in *2021 Reconciling Data Anal., Autom., Privacy, Secur.: Big Data Chall. (RDAAPS)*, Hamilton, Canada, May 18–19, 2021, vol. 159, pp. 1–12. doi: 10.1109/RDAAPS48126.2021.9452002.

[17] A. Rahali, A. H. Lashkari, G. Kaur, L. Taheri, F. Gagnon and F. Massicotte, "DIDroid: Android malware classification and characterization using deep image learning," in *10th Int. Conf. Commun. Netw. Secur. (ICCNS2020)*, Tokyo, Japan, Nov. 27–29, 2020, pp. 70–82.

[18] X. Lu, J. Zhao, S. Zhu, and P. Lio, "SNDGCN: Robust android malware detection based on subgraph network and denoising GCN network," *Expert. Syst. Appl.*, vol. 250, no. 6, pp. 123922, Sep. 15, 2024. doi: 10.1016/j.eswa.2024.123922.

[19] M. Aamir *et al.*, "AMDDLmodel: Android smartphones malware detection using deep learning model," *PLoS One*, vol. 19, no. 1, pp. e0296722, Jan 2024. doi: 10.1371/journal.pone.0296722.

[20] M. Waheed and S. Qadir, "Effective and efficient android malware detection and category classification using the enhanced KronoDroid dataset," *Secur. Commun. Netw.*, vol. 2024, pp. 1–13, Apr. 2024. doi: 10.1155/2024/7382302.

[21] A. Wajahat *et al.*, "Securing android IoT devices with GuardDroid transparent and lightweight malware detection," *Ain Shams Eng. J.*, vol. 15, no. 5, pp. 102642, May 2024. doi: 10.1016/j.asej.2024.102642.

[22] A. Wajahat *et al.*, "Outsmarting android malware with cutting-edge feature engineering and machine learning techniques," *Comput. Mater. Contin.*, vol. 79, no. 1, pp. 651–673, 2024. doi: 10.32604/cmc.2024.047530.

[23] S. Mahdavifar, D. Alhadidi and A. A. Ghorbani, "Effective and efficient hybrid android malware classification using pseudo-label stacked auto-encoder," *Journal of Network and Systems Management*, vol. 30, no. 1, pp. 1–34, Nov. 2021. doi: 10.1007/s10922-021-09634-4.

[24] K. Hamandi, A. Chehab, I. H. Elhajj, and A. Kayssi, "Android SMS malware: Vulnerability and mitigation," in *27th Int. Conf. Adv. Inf. Netw. Appl. Workshops*, Barcelona, Spain, Mar. 25–28, 2013, pp. 1004–1009.

[25] G. Iadarola, F. Martinelli, F. Mercaldo, and A. Santone, "Formal methods for android banking malware analysis and detection," in *Sixth Int. Conf. Internet of Things: Syst., Manage. Secur. (IOTSMS)*, Granada, Spain, Oct. 22–25, 2019, pp. 331–336. doi: 10.1109/IOTSMS48152.2019.8939172.

[26] S. Sayad "Decision tree," Accessed: Feb. 20, 2024. [Online]. Available: https://www.saedsayad.com/decision_tree.htm

[27] F. Salo, A. B. Nassif, and A. Essex, "Dimensionality reduction with IG-PCA and ensemble classifier for network intrusion detection," *Comput. Netw.*, vol. 148, no. 11, pp. 164–175, 2019. doi: 10.1016/j.comnet.2018.11.010.

[28] P. Valdiviezo-Diaz, F. Ortega, E. Cobos, and R. Lara-Cabrera, "A collaborative filtering approach based on naïve bayes classifier," *IEEE Access*, vol. 7, pp. 108581–108592, 2019. doi: 10.1109/ACCESS.2019.2933048.