INTERNATIONAL RESEARCH JOURNAL IN ADVANCED ENGINEERING AND TECHNOLOGY (IRJAET) E - ISSN: 2454-4752 P - ISSN : 2454-4744
VOL 9 ISSUE 6 (2023) PAGES 25-34
RECEIVED:05.11.2023 PUBLISHED:30.12.2023

Dec 30, 2023

A NOVEL MACHINE LEARNING APPROACH FOR ANDROID MALWARE DETECTION BASED ON THE CO-EXISTENCE OF FEATURES

V. VISHAL[1], M. ANITHA[2]

[1] Students, Department of CSE, KINGSTON ENGINEERING COLLEGE, VELLORE - 632 059, Tamil Nadu, India
[2] Assistant Professor, Department of CSE, KINGSTON ENGINEERING COLLEGE, VELLORE - 632 059, Tamil Nadu, India

ABSTRACT

This paper proposes a machine learning model based on the co-existence of static features for Android malware detection. The proposed model assumes that Android malware requests an abnormal set of co-existed permissions and APIs in comparing to those requested by benign applications. To prove this assumption, the paper created a new dataset of co-existed permissions and API calls at different levels of combinations, which are the second level, the third level, the fourth level and the fifth level. The extracted datasets of co-existed features at different levels were applied on permissions only, APIs only, permissions and APIs, and APIs and APIs frequencies. To extract the most relevant co-existed features, the frequent pattern growth (FP-growth) algorithm, which is an association rule mining technique, was used. The new datasets were extracted using Android APK samples from the Drebin, Malgenome and MalDroid2020 datasets. To evaluate the proposed model, several conventional machine learning algorithms were used. The results show that the model can successfully classify Android malware with a high accuracy using machine learning algorithms and the co-existence of features. Moreover, the results show that the achieved classification accuracy depends on the classifier and the type of co-existed features. The maximum accuracy, which is 98%, was achieved using the Random Forest algorithm and the co-existence of permissions features at the second combination level. Furthermore, the results show that the proposed approach outperforms the state-of-the-art model. Using Malgenome dataset, the proposed approach achieved an accuracy of about 98%, while the state-of-the-art achieved an accuracy of about 87%. In addition, the experiments show that using the Drebin dataset, the proposed approach achieved an accuracy of about 95%, while the state-of-the-art achieved an accuracy of about 93%.

1. INTRODUCTION

Smartphone market is growing immensely. According to the ICD report [1], it is estimated that by 2024, the annual sales of mobile phones will reach more than 351 million units globally. Among the several mobiles operating systems, Android is the dominant operating system with over 2.5 billion active users across over 190 countries [2]. The wide range of capabilities offered by smartphones and the rising number of activities carried out by their users, including social networking, online banking, and gaming, has given rise to very serious concerns about device security and personal privacy. Since Android is an open-source platform, it is easy for malware developer to launch their attacks and develop Android malware apps that pose severe harm. Obviously, the impact of Android malware is rising in modern society [3], [4]. Mobile malware is constantly updated with new features to evade detection by anti-malware scanners. Android malware applications usually use three types of breakthrough techniques to get access to the user's devices.

1) Repackaging: is one of the common techniques used to install malware applications. This approach misuses popular applications as the developers install popular applications, disassemble them, inject their malicious code, re-assemble them, and upload the injected app to the third party to be downloaded by users.

2) Update: The developer here may still be using repackaging, but instead of closing the malicious code to the app, they set an update component that even downloads malicious code at runtime.

3) Downloading: The most common technique where the developer encourages users to download their apps is by attracting them to download interesting and useful apps. However, these apps are malicious and may harm their devices.

2. RELATE WORK

Many machine learning approaches have been developed to prevent or detect Android malware. Machine learning algorithms use static, dynamic and hybrid features for static, dynamic and hybrid analysis, respectively. The following subsections discusses some related work that used different types of machine learning approaches, and Table 1 compares between the related work. We should mention here that these algorithms score each feature alone. In the contrary, this paper relies on the co-existence of features to detect Android malware. To the best of our knowledge, the proposed work in this paper is the first work that focuses on the co-existence of features at different levels, and uses this variety of combinations.

A. STATIC BASED APPROACHES:

Using static features to detect Android malware have been studied by many authors. Table 1 compares between the discussed related work in this section. Tiwari and Shukla [10] proposed a machine learning method using logistic regression to detect Android malware based on permissions and API features. The authors tested the model using the full features dataset and a reduced features dataset that consists of 131 features. The authors claimed that their model achieved an accuracy of 97.25% and 95.87% using full features and reduced features datasets respectively. Potha et al.

B. DYNAMIC AND APPROACHES:

This section discusses some related work about using dynamic features in detecting Android malware. Table 1 compares between the discussed related work in this section. Afonso et al. [22] proposed a model based on dynamic analysis of Android applications. The proposed system dynamically detects malware using Android API calls and system call traces. The authors used different datasets from Malgenome project [17] and Virusshare [14] with a total of 7520 apps. The proposed model tested several classification algorithms and achieved an accuracy of 96.66%. However, their approach failed to monitor the malicious behavior in some cases. For example, when an app fails to connect to the Internet, the model stops executing the app without gathering information about any malicious actions.

C. HYBRID BASED APPROACHES:

Unlike the approaches discussed in the previous two subsections, some authors relied on using hybrid approaches that combines both static and dynamic features to detect Android malware. Table 1 compares between some hybrid approaches that are discussed in this subsection. Kouliaridis et al. [4] proposed a tool, called Androtomist, that can apply static and dynamic analysis of Android applications. Androtomist is available as open source software and has two modes: novice and expert users. The authors tested Androtomistusing three Android datasets and several machine learning classifiers.

TABLE 1. Comparison between some related work

Work	Year	Analysis	Feature(s)	Dataset(s)	Algorithm	Accuracy
[19]	2018	Static	API calls	Google Paly[27], Virusshare[14], Contagio [16]	DT	F1 score of 98.24%
[25]	2019	Dynamic	Network traffic	Drebin[13]	C4.5	98.24%
[4]	2020	Hybrid	API calls, Permissions, Intents, Network traffic, Java classes, Inter-process communication	Drebin[13], VirusShare[14], AndroZoo[12]	LR Naive Bayes Random Forest k-NN AdaBoost SGD SVM Ensemble	Drebin: 100% VirusShare:100% AndroZoo: 91.8%
[11]	2020	Static	Permissions, Intents	Drebin[13], VirusShare[14], AndroZoo[12]	extrinsic ensemble method with different base models	Drebin: 93.8% VirusShare: 94.5% AndroZoo:97.3% Mixed: 92.5%
[28]	2020	Hybrid	Permissions, Intents, API Calls, Actions/Events	McAfee[29]	Deep Learning with a state-based input generation approach and the state-of-the-practice popular Monkey tool (stateless method)	97.8% detection rate (with dynamic features only) and 99.6% detection rate (with dynamic + static features)
[15]	2020	Static	API calls, intent, and permissions	Drebin[13], Contagio[16], MalGenome[17]	Hamming FNN,ANN,WANN, KMNN	Varies between 90%-99% according to the type of features used
[18]	2021	Static	raw opcodes, permissions and API calls,	Drebin[13]	Discriminative Adversarial Network (DAN)	F-score of 97.3%
[2]	2021	Static	Permissions, Intents	AndroZoo[12]	AdaBoost, k-NN, LR, NB, MLP, SGD, SVM,RF	91.7%
[10]	2018	Static	Permissions API calls	PRAGaurd and Google Play[27]	LR	97.25%
[30]	2019	static	Permissions Components API calls Network addresses	Drebin[13]	K-NN	99.48%
[31]	2018	static	Permissions Intent filters API calls Constant strings	Drebin[13], Chinese app markets	CNN	97.4%
[32]	2020	static	Opcodes	AMD [19], Drebin[13] VirusShare[14]	BiLSTM LSTM CNN DBN	99.9%
[24]	2019	Dynamic	method calls and inter-component communication (ICC) Intents	Genome[17], Drebin[13], VirusShare[14], AndroZoo[12]	Droidcat	97%
[22]	2015	Dynamic	API calls and system call	Malgenome[17], VirusShare[14]	RF, J.48, SimpleLogistic, NB, SMO, BayesNet, IBK	96.66%
[23]	2016	Dynamic	system calls, Binder communication	Drebin[13]	SVM	94%
[33]	2018	Hybrid	Class Structure, App Names, File Operations, Certificate Metadata, Network Activity, Manifest Metadata, Intent Receivers, Data Leaks, Dynamic Code Loading, Used/Required Permissions, Phone Activity, Crypto Operations	Private dataset	SVM, linear classifier	98.24%
[34]	2018	Hybrid	Sys Calls, SMS, Critical API, User Activity, App Metadata	Genome [17], Contagio[16], VirusShare[14]	K-NN, LDC, QDC, MLP, PARZC, RBF	96.9%
[35]	2016	Hybrid	Permission, Sensitive API Call, sequences, dynamic behaviors	Google Play[27], Contagio[16]	StormDroid (SVM, C4.5, MLP, NB, IBK, Bagging predictor)	93.80%

3.DATA PROCESSING

1) HANDLING IMBALANCED DATASETS

This paper used the random under-sampling method before generating the combinations and during the classification process. The combinations construction must rely on roughly the same number of malware and benign samples to get a reliable system, and build it using real data that can distinguish malware from benign applications. The random undersampling technique was used because it is a simple method for removing samples without imposing any constraints on the data. Furthermore, several experiments have shown that the accuracy does not greatly change in every experiment when using random under-sampling.

2) APK DECOMPILING

The Manifest file, the classes.dex file and the raw resources like images and layout files are all packed in an Android APK file. Dex2jar [11] and ApkTool [28] are two open-source APK decompiling tools used in this work. The ApkTool has been chosen because it is powerful and simple to implement. These tools process one APK file at a time. Therefore, a python code script has been developed to decompile all APK files at the same time. Some APKs cannot decompile due to errors in the file extension by the developer, therefore, they were removed. After removing these files, 2081 APK files were decompiled from the CIC_MalDroid2020 dataset [36]. Then, their Manifest files and smali files were used for the extraction of permissions and static APIs.

4.THE CO-EXISTENCE METHOD:

The proposed approach assumes that a malware requests a unique set of co-existed permissions and APIs, which are different from those requested by benign Android applications. Therefore, this paper constructed different datasets for the co-existing of the two types of features (Permissions, API calls) at different combination levels (level2, level3, level4, level5). The following subsections explain this process.

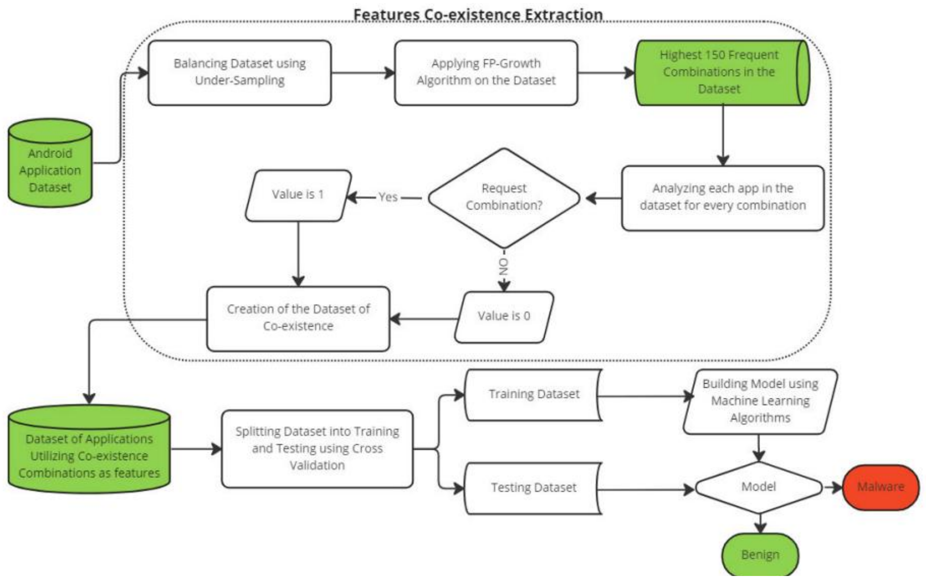
1.FREQUENT PATTERNS EXTRACTION

The effectiveness of frequent pattern-based classification is discussed in [15].The authors showed that the infrequent patterns may disturb the model due to their limited discriminative power. Therefore, they emphasized on the importance of selecting a reasonable minimum support threshold. Moreover, they explained that a frequent

pattern has two properties: every pattern is a combination of single features, and they are frequent. Using the combination of features converts the feature space into a non-linear feature combination, which increases the new feature space expressive power. We believe that the feature co-existence technique captures more underlying semantics than single features, and reduces the number of base features that depend on them to detect malware. Moreover, we realize that the combinations of permissions and APIs can be effective in detecting malicious apps.

2) FP-GROWTH

Data mining helps in extracting information from a dataset to identify patterns and meaningful data. The association rule data mining technique tends to find interesting patterns in transactional data. Two main techniques are used in association rule generation: FP-growth and Apriori algorithms. Agrawal et al. [18] proposed a fast algorithm for mining association rules called the FP-growth method, which is used in this paper. FP-growth is a divide and conquer strategy that consists of two steps: building an FP-Tree and extracting the frequent itemset from the FP-Tree without considering candidates generations.


Features Co-existence Extraction
Android Application Dataset
Balancing Dataset using Under-Sampling
Applying FP-Growth Algorithm on the Dataset
Highest 150 Frequent Combinations in the Dataset
Analyzing each app in the dataset for every combination
Request Combination?
Yes
Value is 1
NO
Value is 0
Creation of the Dataset of Co-existence
Dataset of Applications Utilizing Co-existence Combinations as features
Splitting Dataset into Training and Testing using Cross Validation
Training Dataset
Testing Dataset
Building Model using Machine Learning Algorithms
Model
Malware
Benign

In this research, the FP-growth association rule mining technique was applied to a dataset of binary and single features to extract frequent patterns at different feature set sizes and to use them as co-existing features in classification. The FP-growth algorithm is used to extract frequent patterns at different set of levels based on the frequency of these features in datasets. Several experiments were conducted using different number of top-frequent patterns (50, 75, 100, 150) to select the best number of combinations to be used as a feature in the new conditional dataset. 150 combinations at all levels for all datasets were selected, as this number of combinations achieved the best detection accuracy. Figure 1 shows the system flow from balancing the dataset, extracting co-existence combinations, building a new dataset based on highly frequent patterns as a feature, and then applying machine learning models for classification.

3) THE LEVELS OF FEATURES CO-EXISTENCE

Four levels of features co-existence are considered in this paper, which are level 2, level 3, level 4, and level 5. To clarify this process, suppose that the dataset consists of five APK samples and the extracted permissions features are (READ_SMS, READ_PHONE_STATE, WRITE_SMS, INTERNET, and PHONE_CALL). Table 2 shows an example of the co-existence of permissions at every level. Table 3 shows an example of the co-existence of permissions features in the dataset APKs at level 2. The values in the dataset are based on the analysis of feature set requests by the apps. If the app requests a feature set in a given column, the value is 1, otherwise it is 0. For example, the first column value is 1 if apps request (READ_SMS and SEND_SMS), otherwise the value is 0. This dataset is fed into machine learning algorithms to build a model that can differentiate malware from benign applications based on highly co-existed features.

TABLE 2. Examples of permissions co-existence at different levels

Levels	An Example of Features Co-existence
Level2	READ_SMS, READ_PHONE_STATE
Level3	READ_SMS, READ_PHONE_STATE, WRITE_SMS
Level4	READ_SMS, READ_PHONE_STATE, WRITE_SMS, INTERNET
Level5	READ_SMS, READ_PHONE_STATE, WRITE_SMS, INTERNET, and PHONE_CALL

TABLE 3. An example of the permissions co-existence at level 2 in the dataset

APPS/Permissions	Read sms - send sms	Read phone state -write external storage	Internet -receive sms	class
APP1	0	1	1	0
APP2	1	1	0	1
APP3	1	0	1	1
App4	0	0	1	1
App5	1	1	0	0

A. SELECTING THE TOP FREQUENT CO-EXISTED FEATURES:

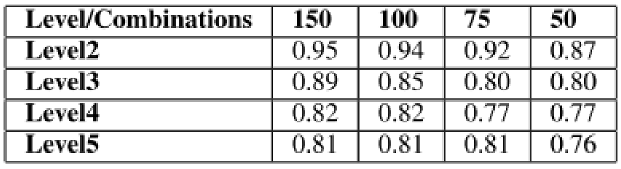
Highly frequent candidates will not be the best discriminative patterns as they appear in a large portion of the dataset in different classes. Similarly, very low-frequent candidates may disrupt the model's accuracy due to overfitting. The best co-existence combination is the one that is highly frequent in malware and low or non-frequent in benign applications and vice versa.

The proposed technique extracts the top 150 frequent feature combinations (co-existence), using FP-Growth, at different levels that are requested by applications frequently, as discussed in third section. The models were built on these combinations as features to measure their effectiveness in classifying malware from benign applications. The model chose this number of top features combinations since it achieved the best accuracy results; the proposed model tested the classifiers algorithms using different number of combination features, which are 50, 75, 100 and 150, from the different combinations of features co-existence extracted from CIC_MalDroid2020 dataset. The results using Random Forest are shown only in this paper since Random Forest achieved the best accuracy as will be shown in subsequent sections and since the experiments on the other classifiers lead to the same conclusion. The results are shown in in Tables 4, 5 and 6.

B. RESULTS USING CIC_MALDROID2020 DATASET:

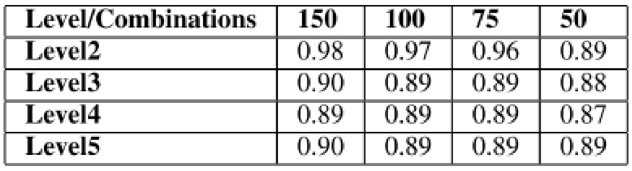
This section discusses the results of testing the machine learning algorithms using different features co-existence, which were extracted from CIC_MALDROID2020 Dataset. The experiments were conducted at the different levels of co-existence 2,3,4 and 5. Figure 2, Figure 3 and Figure 4 show the results of these experiments. Figure 2 shows the results of testing the machine learning algorithms using the permissions features only at different

TABLE 4. Accuracy of the random forest classifier for API co-existence in the CIC_MALDROID2020 dataset at different co-existence levels and different number of top relevant combinations.



Level/Combinations	150	100	75	50
Level2	0.95	0.94	0.92	0.87
Level3	0.89	0.85	0.80	0.80
Level4	0.82	0.82	0.77	0.77
Level5	0.81	0.81	0.81	0.76

TABLE 5. Accuracy of the random forest classifier for permission co-existence in the CIC_MALDROID2020 dataset at different co-existence levels and different number of top relevant combinations.



Level/Combinations	150	100	75	50
Level2	0.98	0.97	0.96	0.89
Level3	0.90	0.89	0.89	0.88
Level4	0.89	0.89	0.89	0.87
Level5	0.90	0.89	0.89	0.89

TABLE 6. Accuracy of the random forest classifier for permission-API co-existence in the CIC_MALDROID2020 dataset at different co-existence levels and different number of top relevant combinations.

levels of co-existence. As shown in the figure, Random Forest, Decision Trees and SVM algorithms achieved the best and similar results at all co-existence levels; all of them achieved the highest accuracy, which is about 98%. Moreover, all algorithms achieved the best accuracy at the second level of co-existence. Similarly, Figure 3 shows the results of testing the machine learning algorithms using API features only at different levels of co-existence. The best accuracy, which is 95%, was achieved by Random Forest, Decision Trees and SVM algorithms at level 2. Obviously, the achieved accuracy using API features only is less than the accuracy achieved by the same algorithms when using permissions only co-existence. The results of testing machine learning algorithms using permissions-API features at different levels of co-existence are shown in Figure 4. As shown in the figure, the best accuracy, which is 98%, was achieved by the same algorithms at level 2.

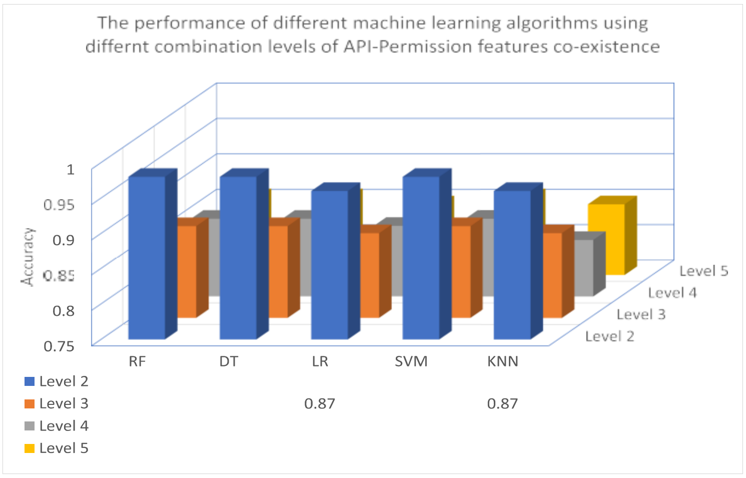


Fig (a)

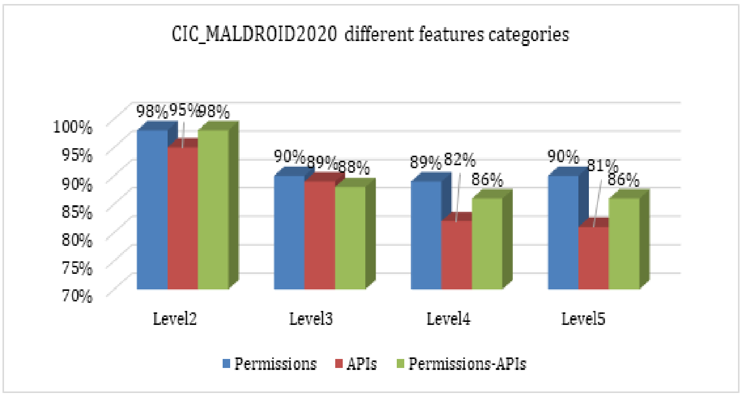


Fig (b)

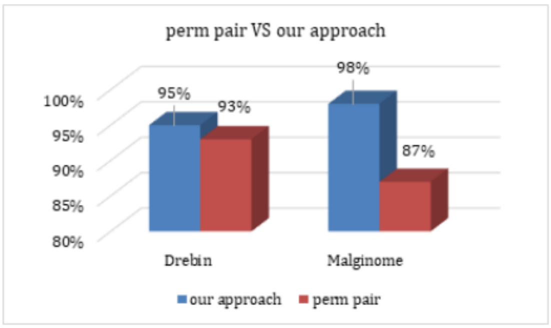


Fig (c)

FIGURE 16. Accuracy comparisons between the proposed approach and the PermPair method.

approach in this paper utilizes the FP-growth method to analyze the co-existence of features from level 2 to level 5. Furthermore, the proposed approach considers different feature categories, permissions, APIs, and API frequency when determining the best feature category for co-existence-based detection using machine learning techniques. The PermPair used the scoring technique to detect Android malware using Drebin and Malgenome datasets. Meanwhile, the proposed approach achieved its best accuracy using Random Forest algorithm. Table 7 shows the size of the common datasets in the proposed work and the Perm-Pair technique. Figure 16 shows the comparisons between both approaches using the same datasets at the second level of features co-existence. The results show that the proposed approach achieved an accuracy of about 98% and 95% using Malgenome and Drebin datasets respectively, while the PermPair technique achieved an accuracy of about 87% and 93% using the same datasets, respectively.

5. CONCLUSION AND FUTURE WORK

The paper has proposed a novel approach for detecting Android malware using the FP-growth algorithm, which is an association rule mining technique that is used to extract the frequent patterns of features at different feature co-existence levels. Moreover, the paper has created three datasets of co-existed features, which are co-existed permissions features only, co-existed API features only and co-existed permissions and API features. Furthermore, the paper has created the features co-existence at four levels, which are level two, level three, level four and level five. To test the proposed approach, several machine learning algorithms were used, which are Random Forest, Decision Trees, Logistic Regression, SVM and KNN. The experiments have shown that Random Forest, DT and SVM algorithms achieved the best accuracy of about 98% at level 2 of co-existence using permission-API co-existence in the CIC_MALDROID2020 dataset. Furthermore, the experiments have shown that all machine learning algorithms achieved the best results at the second level of features co-existence. Moreover, the experiments have shown that using the frequent API co-existence is better than using API features in Android malware detection. That is, extracting API frequencies from each APK, converting frequencies to existence based on an average of API call frequencies as a threshold, and then applying the co-existence technique achieved better accuracy than using API co-existence without considering API calls frequencies. In addition, the paper has compared the proposed approach with the state-of-the-art model PermPair and has shown that the proposed approach has outperformed the state-of-the-art model (PermPair), which achieved an accuracy of about 93% and 87% using Drebin and Malgenome datasets, respectively. Meanwhile, the proposed approach Achieved an accuracy of about 95% and 98% using the same datasets. The results have shown that using the features co-existence is an effective method to detect Android. As a future work, we plan to evaluate the features co-existence approach using dynamic features.

6.REFERENCES

[1] H. Menear. (2021). IDC Predicts Used Smartphone Market Will Grow 11.2% by 2024. Accessed: Oct. 30, 2022. [Online]. Available: https://mobile-magazine.com/mobile-operators/idc-predicts-usedsmartphone-market-will-grow-112-2024?page=1

[2] D. Curry. (2022). Android Statistics. Accessed: Oct. 30, 2022. [Online]. Available: https://www.businessofapps.com/data/android-statistics/

[3] O. Abendan. (2011). Fake Apps Affect Android Os Users. Accessed: Oct. 30, 2022. [Online]. Available: https://www.trendmicro.com/ vinfo/us/threat-encyclopedia/web-attack/72/fake-apps-affect-android-osusers

[4] C. D. Vijayanand and K. S. Arunlal, ''Impact of malware in modern society,'' J. Sci. Res. Develop., vol. 2, pp. 593–600, Jun. 2019.

[5] M. Iqbal. (2022). App Download Data. Accessed: Oct. 30, 2022. [Online]. Available: https://www.businessofapps.com/data/app-statistics/

[6] K. Allix, T. Bissyand, Q. Jarome, J. Klein, R. State, and Y. L. Traon, ''Empirical assessment of machine learning-based malware detectors for android,'' Empirical Softw. Eng., vol. 21, pp. 183–211, Jun. 2016.

[7] Y. Zhou and X. Jiang, ''Dissecting Android malware: Characterization and evolution,'' in Proc. IEEE Symp. Secur. Privacy, May 2012, pp. 95–109.

[8] J. Scott. (2017). Signature Based Malware Detection is Dead. Accessed: Oct. 30, 2022. [Online]. Available: https://icitech.org/wpcontent/uploads/2017/02/ICIT-Analysis-Signature-Based-MalwareDetection-is-Dead.pdf

[9] Q.M. Y. E. Odat. Accessed: Dec. 27, 2022. [Online]. Available: https://github.com/esraa-cell28/a-novel-machine-learning-approach-forandroid-malware-detection-based-on-the-co-existence

[10] S. R. Tiwari and R. U. Shukla, ''An Android malware detection technique based on optimized permissions and API,'' in Proc. Int. Conf. Inventive Res. Comput. Appl. (ICIRCA), Jul. 2018, pp. 258–263.

[11] (2018). Dex2jar—Tools To Work With Android.dex&Java.Class Files. Accessed: Oct. 30, 2022. [Online]. Available: https://kalilinuxtutorials.com/dex2jar-android-java/

[12] Androzoo. Accessed: Jul. 30, 2022. [Online]. Available: https://androzoo.uni.lu/

[13] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, ''Drebin: Effective and explainable detection of Android malware in your pocket,'' in Proc. NDSS, Feb. 2014, pp. 23–26.

[14] Virusshare. accessed: Jul. 30, 2022. [Online]. Available: https://virusshare.com/

[15] H. Cheng, X. Yan, J. Han, and C.-W. Hsu, ''Discriminative frequent pattern analysis for effective classification,'' in Proc. IEEE 23rd Int. Conf. Data Eng., Apr. 2007, pp. 716–725.

[16] M. Parkour. Contagio Mini-Dump. accessed: Jul. 30, 2022. [Online]. Available: http://contagiominidump.blogspot.it/