

Received 27 December 2022, accepted 6 February 2023, date of publication 13 February 2023, date of current version 17 February 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3244656

RESEARCH ARTICLE

A Novel Machine Learning Approach for Android Malware Detection Based on the Co-Existence of Features

ESRAA ODAT¹ AND QUSSAI M. YASEEN^{1,2,3}

¹Department of Computer Information Systems, Jordan University of Science and Technology, Irbid 22110, Jordan

²Artificial Intelligence Research Center AIRC, College of Computer Engineering and Information Technology, Ajman University, Ajman, United Arab Emirates

³Faculty of Computer and Information Technology, Jordan University of Science and Technology, Irbid 22110, Jordan

Corresponding author: Qussai M. Yaseen (q.yaseen@ajman.ac.ae)


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%.

INDEX TERMS Android, co-existence, FP-growth, machine learning, malware.

I. 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,

The associate editor coordinating the review of this manuscript and approving it for publication was Diana Gratiela Berbecaru .

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.

The Google Play Store, which is the marketplace for Android applications, has more than 3.43 million apps as of January 2021 [5]. Many third-party and non-Google stores are appearing, such as AppBrain and AppChina, which provide applications to be downloaded by users. The applications provided by the third-party markets have a high risk of being malicious apps that are not monitored or detected. Allix et al. [6] reported that 22% of Google Play applications were identified as malicious applications and 50% of AppChina applications were recognized as malicious apps.

Many defense techniques have been used to detect and prevent malicious applications. Signature-based malware detection is an early method that works as a pattern comparison between already exposed apps and new apps. It works well for known malware but it is easily evaded by obfuscation mechanisms or unknown malware. To evaluate the effectiveness of existing signature-based anti-malware scanners, Zhou and Jiang [7] tested four different mobile security applications against more than 1200 Android applications. The results showed that the existing anti-malware apps cannot detect obfuscated or repackaged malware apps. Similarly, Scott [8] applied an obfuscation technique to ten different malware applications from different families. The results showed that none of them can detect malicious applications after obfuscation.

Machine learning methods are applied to detect Android malware and distinguish them from benign ones without comparing the patterns of the known Android malware. Machine learning methods work smartly by building a model based on sample data, known as “training data”, to make predictions or decisions without being explicitly programmed. The malware detection techniques using machine learning algorithms are classified into two types, which are static analysis and dynamic analysis. In static analysis, an Android application is examined without running it. In contrast, in dynamic analysis, the application is run in a controlled environment to analyze its behavior.

Static Analysis uses static features that are extracted from the manifest (i.e. permissions requested by apps), the source code (i.e. API calls), and intents. Many of the extracted features may be disruptive features. For example, many permissions are requested by both benign and malware applications.

Therefore, different feature selection methods are applied to select the most relevant features that accurately classifies malware apps. Predefined feature selection algorithms utilize statistical methods to score the correlation or dependency between input variables and output or class variables. Most of these algorithms score each feature alone. In the contrary, this paper relies on the co-existence of features to detect Android malware. The contributions of this paper are summarized as follows.

- 1) The paper extracted new datasets of co-existed features using the APK samples gathered from three different datasets, which are Drebin, Malgenome, and MalDroid2020. The new dataset is available online for research use [9].
- 2) The paper applied the Association rule mining technique (FP-growth) to construct a novel co-existence-based dataset that depends on the existing of different features together.
- 3) The paper extracted the features co-existence of permissions only, APIs only, permissions and APIs, and APIs and APIs frequencies.
- 4) The paper extracted the features co-existence at different levels, which are level 2 (combinations of two features), level 3 (combinations of three features), level 4 (combinations of four features), and level 5 (combinations of five features).
- 5) The paper used several machine learning algorithms to test the effectiveness of the proposed approach.
- 6) The experiments show that the proposed model can successfully classify Android malware with a high accuracy of about 98% using the RF classifier and the co-existence of permissions features at the second combination level. Moreover, the results show that the proposed approach outperforms the state-of-the-art model.

The rest of the paper is organized as follows. The next section discusses some related work. Section III discusses the methodology. Section IV demonstrates and discusses the experiments and results. Finally, Section V concludes the work.

II. 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 discuss 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. [11] proposed an extrinsic random-based ensemble method, called ERBE. The proposed model used an aggregation function that combines the output of all base models and computes the average of the output of the base models instances separately. The authors tested the effect of external instances and showed that ensembles based on a large and homogeneous external instance are effective more than small and heterogeneous external instances. The experimental results on different datasets, namely AndroZoo [12], Drebin [13] and Virusshare [14], showed that the proposed approach achieved an accuracy up to 97.3% using AndroZoo dataset. Similarly, using AndroZoo dataset, Kouliaridis et al. [2] tested the applicability of Principal Component Analysis (PCA) and t-distributed stochastic neighbor embedding (t-SNE) methods in Android malware detection. They tested several base prominent classifiers and how they can be used together to build an accurate ensemble, and proposed a simple ensemble method and a complex ensemble method. They claimed that their method is effective of assembling using AndroZoo benchmark corpora. In addition, they claimed that ensembles using a heterogeneous and large set of base models provide a stable performance. The results showed that the accuracy using the fusion function AVERAGE(AVG) achieved an accuracy of 91.7%.

Taheri et al. [15] proposed four Android malware detection methods using Hamming distance, which are First Nearest Neighbors (FNN), All Nearest neighbors (ANN), Weighted All Nearest Neighbors (WANN), and k-medoid based nearest neighbors (KMNN). The authors tested their methods using three datasets, which are Drebin [13], Contagio [16], and Genome [17]. The authors claimed that the proposed methods using different types of features of API, intent, and permission features, achieved an accuracy up to 99%.

Millar et al. [18] proposed an Android malware detection model based on Discriminative Adversarial Network (DAN), called DANdroid. The authors claimed that their work has three contributions. The first contribution is the capability of their approach in discriminating adversarial learning results in malware feature representations. The second one is the using of three feature sets (raw opcodes, permissions and API calls) in increasing the obfuscation resilience in a multi-view deep learning architecture. The third one is the capability of their approach in generalizing over rare and future obfuscation approaches that do not exist in the training.

Furthermore, the authors claimed that their model achieved an average F-score of 97.3% using Drebin dataset [13].

Tao et al. [19] studied Android apps searching for hidden patterns of malware highly sensitive APIs. The authors implemented an automated malware detection system, called MalPat. They conducted many experiments using a dataset of 31185 benign apps and 15 336 malware samples they collected from Google play, Virusshare [14] and Contagio [16]. The authors claimed that MalPat achieved an F1-score of 98.24% and outperformed the state-of-the-art methods MUD-FLOW [20], Drebin [13] and DroidAPIMiner [21].

B. DYNAMIC BASED 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.

Dash et al. [23] proposed an approach that uses application runtime behaviors only to classify Android malware into families. The proposed approach used Support Vector Machines SVM and Conformal Prediction. The experiments were conducted using system calls and Binder communication features on Drebin dataset [13] and achieved an accuracy of 94%. However, their approach suffered from the limited information gathered from tracing low-level events and the insufficient coverage when testing applications.

Cat et al. [24] proposed a dynamic classification model for Android Applications, called DroidCat. It used different dynamic features such as intents, method calls, and inter-component communication (ICC). The authors tested Droidcat using a large dataset of about 34343 apps collected from Genome [17], AndroZoo [12], VirusShare [14] and Drebin [13]. The authors claimed that DroidCat achieved an accuracy of about 97%. Moreover, the authors stated that some dynamic features, such as the distribution of method calls over user code and libraries feature capturing app execution structure, are very important in the classification process in comparing to some used features such as sensitive flows.

Wang et al. [25] proposed a framework for Android malware detection using network traffic as features. The authors tested the machine learning algorithm C4.5 using Drebin dataset. The experiments showed that the proposed model achieved an accuracy of 97.89%. Similarly, Sun et al. [26] proposed "Patronus", which is a machine learning model that can dynamically detect Android malware using

runtime information. The authors claimed that Patronus can detect malware accurately without posing high performance overhead or battery consumption.

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 Androtomist using three Android datasets and several machine learning classifiers. In addition, the authors used an ensemble approach by averaging the output and showed the most influencing features. The authors claimed that Androtomist achieved a 100% accuracy on Drebin [13] and VirusShare [14] datasets, while it achieved 91.8% using Androzoo dataset [12].

Alzaylaee et al. [28] proposed a deep learning system, called DL-Droid, to detect Android malware using dynamic analysis and stateful input generation. Moreover, the authors tested the performance of the stateful input generation approach using random-based input generation as a baseline for comparison. The experiments were conducted using a large dataset of more than 30,000 Android benign and malware applications. They showed that DL-Droid achieved a high accuracy up to 97.8% using dynamic features only and 99.6% using both dynamic and static features. The authors argued that there is a high significance for enhanced input generation in dynamic analysis since DL-Droid outperformed the state-of-the-art approaches tested in their work.

Lindorfer et al. [33] proposed a hybrid system, called MARVIN, that uses both static and dynamic analysis to detect unknown Android malware. MARVIN used a rich set of features such as Class Structure, Certificate Metadata, Network Activity, Data Leaks, Dynamic Code Loading, Used/Required Permissions, Crypto Operations, etc. Moreover, the proposed approach was tested using a large dataset that consists of 135,000 Android apps and 15,000 malware samples. The authors claimed that MARVIN achieved an accuracy of 98.24%.

Chen et al. [35] proposed a streaming-based machine learning-based MD model, called StormDroid. StormDroid is a hybrid approach that uses a variety of features such as Permission, Sensitive API Call, sequences, dynamic behaviors to classify Android malware. The proposed model tested different machine learning algorithms such as SVM, C4.5, MLP, NB, IBK, Bagging predictor on a dataset collected from Google Play and Contagio [16] and consisted of about 8,000 applications. The proposed model supports a large-scale and scalable analysis that can monitor static and dynamic behaviors. The authors claimed that StormDroid achieved an accuracy of 93.8% and improved the efficiency

rate by approximately three times in comparing to a single thread.

Saracino et al. [34] proposed a cross-layer machine learning model, called MADAM, which uses different set of static and dynamic features, such as Sys Calls, SMS, Critical API, User Activity and App Metadata. MADAM tested different algorithms, such as K-NN, LDC, QDC, MLP, PARZC, RBF using a large dataset collected from Genome [17], Contagio [16] and VirusShare [14]. The authors claimed that their model achieved an accuracy of 96.6%. Moreover, the authors claimed that their model has limited battery consumption and low performance overhead.

III. METHODOLOGY

The proposed method consists of two parts: preparing a new dataset of co-existed features and using this dataset to classify malware and benign apps using different machine learning classifiers. The datasets of co-existed features consists of permissions only, API Calls only, and a combination of permissions and API calls as well as API frequencies.

A. DATA COLLECTION

The Android APKs used in this work have been obtained from multiple sources to test the effectiveness of the proposed technique. The paper used the APKs from three datasets, which are the Drebin dataset, the Malgenome dataset, and the CIC_MALDROID2020 dataset [36]. The Drebin and Malgenome datasets have been acquired from [37], which provided them as supplementary material. The malware samples of the two datasets named Drebin-215 and Malgenome-215 originate from [13] and [7], respectively. Both datasets contain 215 features, and consist of four feature categories: permissions, APIs, intents, and command signatures, where 181 of them are permissions and APIs. CIC_MALDROID2020 dataset is a recent and large dataset of malware and benign Android application packages (APKs) downloaded from the Canadian Institute for Cybersecurity [36]. Based on these Datasets, this paper generated new datasets of co-existed features and made these datasets available online for research use [9].

1) DREBIN-215 DATASET

The Drebin dataset [13] contains 15031 different applications; 5,550 applications from 179 different malware families and 9477 benign applications. The samples were collected from 2010 to 2012. Using this dataset, the paper extracted 181 different features; 109 permissions and 72 APIs.

2) MALGENOME-215 DATASET

The Malgenome dataset [7] is a malware dataset that was active from 2012 to the end of 2015 and contains 3798 different applications; 1260 applications belong to 49 different malware families; and 2538 benign applications. Using this dataset, the paper extracted 181 features and 109 permissions and 72 APIs.

TABLE 1. A 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 Naïve 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) CIC_MALDROID2020 DATASET

The CIC_MALDroid2020 is provided by the Canadian Institute for Cybersecurity (CIC) [36]. It is a comprehensive educational, research, and enterprise entity that combines researchers from the social sciences, business, computer science, engineering, law, and science to share creative ideas. The dataset has the following four characteristics: large, recent, diverse, and comprehensive. The institute gathered over 17,341 Android samples from a variety of sources,

including VirusTotal service, the Contagio security blog [38], AMD [19], MalDozer [39], and other datasets used by recent contributions [40], [25] and [39]. The dataset consists of four malware categories, which are Adware, Banking malware, SMS malware, and Riskware. We used the adware family apps as a malicious software sample with benign apps to find the coexistence of features in the same malware category. Adware is an advertisement that is frequently hidden within authorized apps that have been infected by malware.

Adware can infect a mobile phone, causing it to download specific forms of Adware and allowing attackers to steal sensitive data. The used dataset consists of 2081 applications, 995 Adware application, and 1086 benign applications.

B. DATA PREPROCESSING

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 under-sampling 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 [41]. Moreover, the number of examples is enough for training and testing the machine learning models, and each application is decompiled alone. Therefore, eliminating some applications does not lead to a significant loss of data.

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.

3) FEATURES VECTOR AND DATASETS CONSTRUCTION

The samples obtained from the CIC_MALDROID2020 dataset were in APK raw format. Therefore, a Python code developed to extract all requested permissions from manifest files after decompiling APKs. Moreover, using smali files, we extracted static API calls in two forms: existence and frequencies. Next, a permissions-APKs matrix, an APIs-APKs matrix, a permission and API-APKs matrix, and an API frequencies-APKs matrix were constructed. The matrices consist of APKs as instances and permissions and API as features. To clarify this point, let R be a vector containing a set of j number of Android permissions and API calls. For every i th application in the dataset, we generated a binary sequence $R_i = \{r_1, r_2, \dots, r_j\}$, where r_x denotes 1 if the corresponding permission or API exists in the application and 0 if it does not exist. Moreover, the class feature was added to the dataset, where 1 indicates a malware and 0 indicates a benign APK.

4) PERMISSIONS CLEANING

It is necessary to eliminate some extracted permissions from the manifest file due to some reasons. In this work, after decompiling CIC_MALDROID2020 APKs, some extracted permissions were removed because of the following reasons:

- Duplicated permissions.
- Incorrect permissions that are commonly caused by input typos or wrong format.
- Permissions were requested by only one or two applications, adapted for specific system users to be more persuasive to their local users, such as Huawei.

Therefore, from 1633 permissions extracted from CIC_MalDroid2020 samples, only 825 correct, valuable, and frequently used permissions were kept.

C. 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 itemsets from the FP-Tree without considering candidates generations. Each path of the FP-Tree

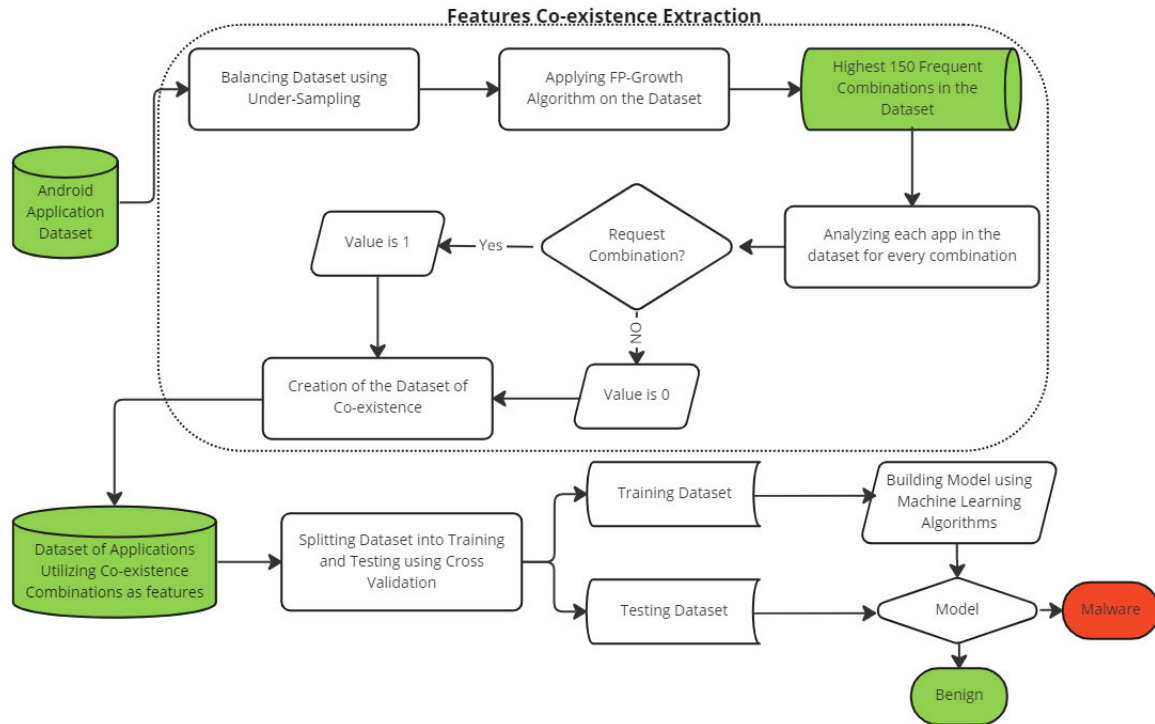


FIGURE 1. The Proposed co-existence-based detection model at every level.

represents frequent itemsets, where nodes in the path are in a decreasing order based on the frequency. It also maintains the association between itemsets. Moreover, Kavitha et al. [42] conducted a comparative analysis study between Apriori and FP-growth. Their work compared the performance of frequent pattern searching in large datasets using FP-growth method that uses divide and conquer and Apriori method that uses the breadth-first search approach. The FP-growth algorithm aims to extract frequent item sets from transactional data without candidate set generation, whereas the Apriori algorithm extracts candidate itemsets through many scans and then filters frequent patterns. FP-growth is fast as it consumes less time in frequent itemset generation than Apriori. Moreover, it is a memory-efficient technique due to its structure, which extracts frequent patterns without requiring candidate generation.

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.

D. CO-EXISTENCE EXTRACTION PREPROCESSING

Preprocessing is necessary for permission-API datasets and API frequency datasets to prepare them for feature co-existence extraction. However, no preprocessing is required for permission-only datasets and API-only datasets.

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

1) PERMISSION-API DATASETS

One of the goals of the proposed method is to measure the effectiveness of relying on the co-existence of a mix of permissions and APIs in the same set to discriminate malware from benign applications. Therefore, the method extracts all co-existing features at a specific level, and those that contain only APIs or only permissions are excluded from the co-existence matrix. Next, the system depends only on highly co-existing features that have mixed features of permissions and APIs. For example, if the dataset contains only permissions combinations such as (READ-SMS, INTERNET), the system will eliminate it because the purpose of this dataset is to construct a model using permission-API combinations.

2) API FREQUENCY DATASETS

In this part, the proposed system considers only APIs with high frequency in each APK. It uses a predefined threshold to consider an API as frequent or not. The technique started by developing an equation that determines the average of frequencies of all requested APIs by all APKs. Next, this average is used as the threshold based on which an API is considered frequent or not. That is, if the frequency of an API is greater than the threshold value, the API is considered frequent and its existence value is set to 1, otherwise, it is not considered frequent and its existence value is set to 0.

To clarify this point, assume that N is the total number of APIs, M is the total number of API calls (total frequencies for all APIs), and A is the threshold value, then:

$$A = N/M$$

where A is computed as the average of API frequencies. That is, A is the minimum number of frequencies for an API to be considered in the API's Co-existence matrix.

Several experiments were conducted using different threshold values, however, this formula achieved the best results. The paper applied this idea using the CIC_MALDROID2020 dataset.

IV. EXPERIMENTS AND RESULTS

The experiments aim at evaluating the performance of the proposed technique using both permissions and API calls features in three forms, which are: permission-only co-existence, API-only co-existence, and permission-API co-existence.

Moreover, the proposed model aims at developing a malware detection method based on the co-existence of a few features. Five machine learning classifiers were tested, which are Random Forest (RF), K-Nearest Neighbors (KNN), Logistic Regression (LR), Decision Tree (J48), and Support Vector Machine (SVM). These algorithms were selected because they are the top most used machine learning algorithms [43]. The results are shown in terms of accuracy since the dataset is balanced. The analysis includes identifying the best model at the best co-existence level which accurately classifies malware and benign apps, finds the correlation between co-existence level and performance, and compares feature sets to generalize which is the best form of features co-existence that well classifies malware applications. The experiments were conducted using the datasets acquired for every subset—permission dataset, API dataset, permission-API dataset and API-frequencies dataset. The results are analyzed and compared for every dataset separately as shown in subsequent sections.

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.

Level/Combinations	150	100	75	50
Level2	0.97	0.93	0.88	0.87
Level3	0.89	0.8	0.81	0.74
Level4	0.82	0.83	0.79	0.79
Level5	0.81	0.82	0.79	0.78

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.

Figure 5 summarizes the best achievements at different levels of features co-existence, which were achieved by Random Forest, Decision Trees, and SVM. The figure shows that the performance of machine learning algorithms degrades when increasing the co-existence level. In addition, it shows that using either permissions only or permission-API features at the second level of co-existence achieves the same accuracy, and outperforms the accuracy of using APIs only. However, it shows that the accuracy degrades sharply when increasing the level of co-existing using either API features only or permission-API only in comparison to using permission features only.

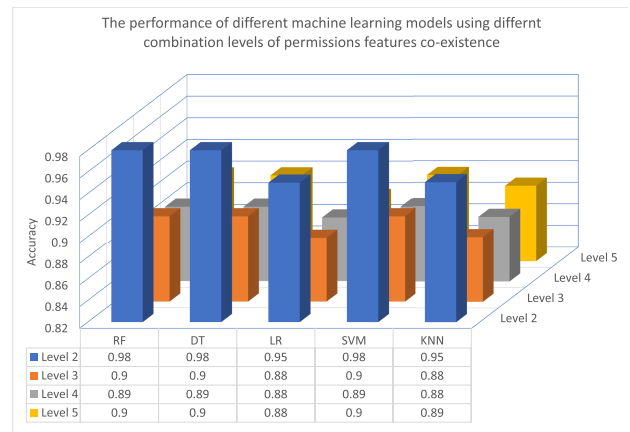


FIGURE 2. The performance of different machine learning algorithms using different combination levels of permissions features co-existence in CIC_MalDroid2020 dataset.

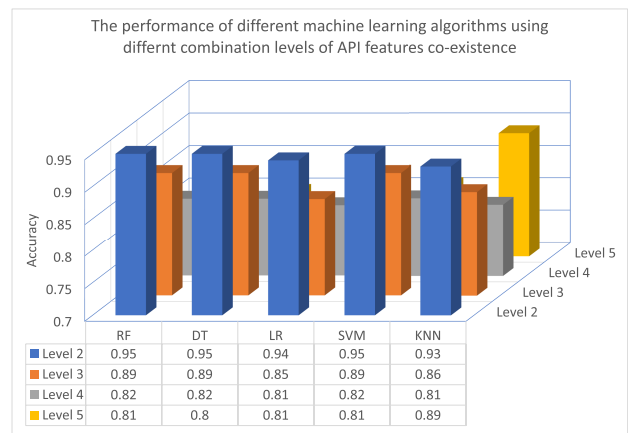


FIGURE 3. The performance of different machine learning algorithms using different combination levels of API features co-existence in CIC_MalDroid2020 dataset.

The figures show that the second level of co-existence-based detection outperforms other levels in all feature categories: permissions, APIs, and permissions-APIs. Therefore, level 2 has discriminative co-existence combinations that differentiate malware and benign applications. At level 2, as the proposed system picks the top k frequent candidates, it drills down to get the specified number of k, so the chance increases to get more information and include combinations with various frequencies, which has high discriminative power. Moreover, as the level increases, the frequency of combination decreases as it exists in a small portion of the dataset. Therefore, the model’s ability to detect malware depends on the discriminative power of co-existence features and on the combinations variations at a specific level.

C. RESULTS USING MALGENOME DATASET

This section discusses the results of testing the machine learning algorithms using different features co-existence that were extracted from Malgenome Dataset. The experiments were conducted at the different levels of co-existence 2,3,4 and 5. The results are shown in Figure 6, Figure 7 and Figure 8.

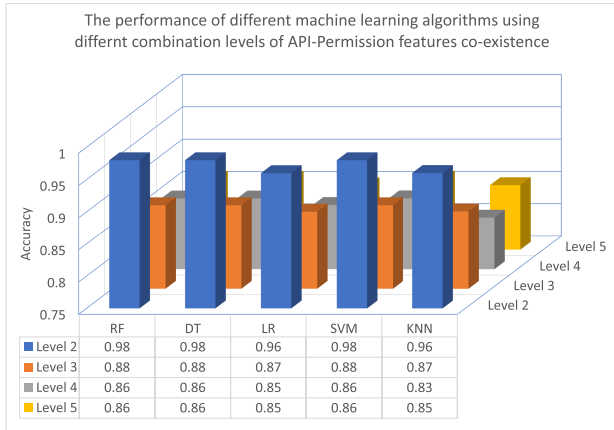


FIGURE 4. The performance of different machine learning algorithms using different combination levels of permission-API features co-existence in CIC_MalDroid2020 dataset.

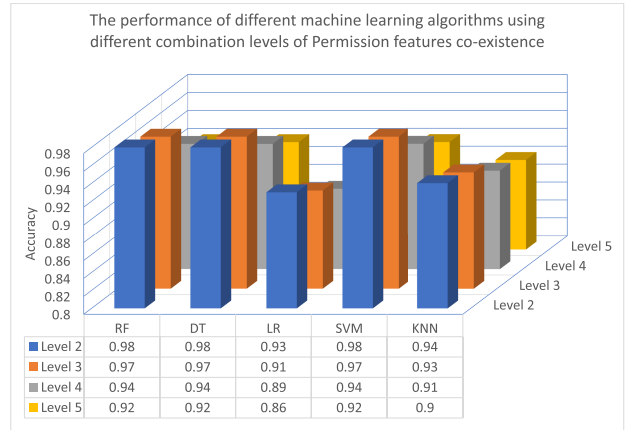


FIGURE 6. The performance of different machine learning algorithms using different combination levels of permissions features co-existence in Malgenome dataset.

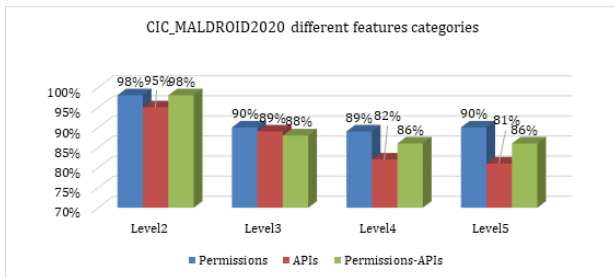


FIGURE 5. RF, DT and SVM accuracy at different feature categories and different levels using CIC_MALDROID 2020.

Figure 6 shows the performance of machine learning algorithms at different co-existence levels using the permissions features only. It shows that Random Forest, Decision Tree and SVM achieved the highest accuracy, which is 98% at level 2. Similarly, the same algorithms achieved the best results when using API features only and permission-API features as shown in Figures 7 and 8 respectively. Using API features only, Random Forest, SVM and Decision tree algorithms achieved an accuracy of about 97%, while using permissions-API combinations enhanced the accuracy of these algorithms to 98%. Figure 9 summarizes the results achieved by Random Forest, Decision Tree and SVM algorithms using Malgenome dataset. Obviously, the best results were achieved at level 2 regardless the used features. The maximum achieved accuracy is when these algorithms used permissions features only or permission-API features at level 2. However, the degradation in accuracy as the features co-existence level increases is maximized when using permission-API features.

D. RESULTS USING DREBIN DATASET

This section discusses the results of testing the machine learning algorithms using different co-existed features that were extracted from Drebin Dataset. Similar to the previous datasets, the experiments were conducted at the different levels of co-existence 2,3,4 and 5. The results are shown in Figure 10, Figure 11 and Figure 12. As in using

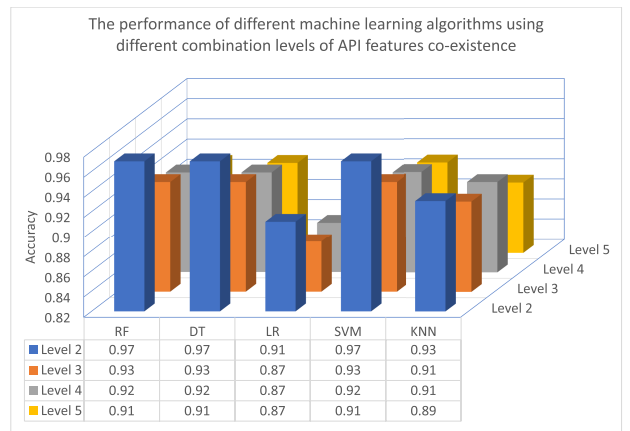


FIGURE 7. The performance of different machine learning algorithms using different combination levels of API features co-existence in Malgenome dataset.

CIC_MalDroid2020 and Malgenom Datasets, using Drebin datasets, the Random Forest, Decision Trees, and SVM algorithms achieved the best results using different co-existed features at all levels of co-existence. However, the best accuracy, which is 97%, was achieved when using the co-existed permission-API features at level 2. This is different than what was achieved in the previous experiments using CIC_MalDroid2020 and Malgenom Datasets, where the best results were achieved using either permissions only or permission-API combinations at level 2.

Figure 13 shows the summary of the performance of RF, DT and SVM algorithms, which achieved the best results at different levels of features co-existence. The figure shows that the degradation in performance of the algorithms increases as the level of combination (features co-existence) increases, which is similar to the results achieved in previous sections.

E. API FREQUENCY

The previous experiments used the existence of API features in creating the co-existence datasets. However, they

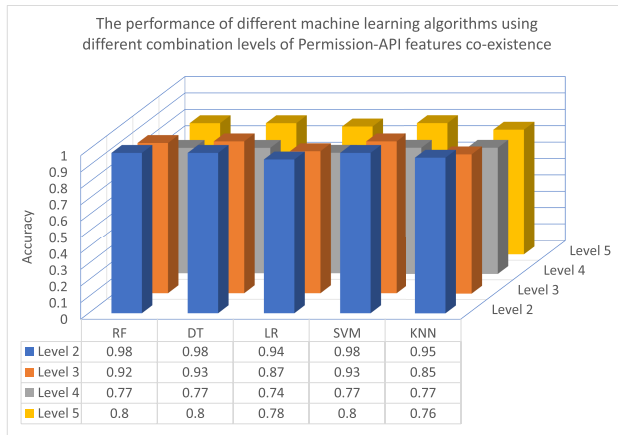


FIGURE 8. The performance of different machine learning algorithms using different combination levels of permission-API features co-existence in malgenome dataset.

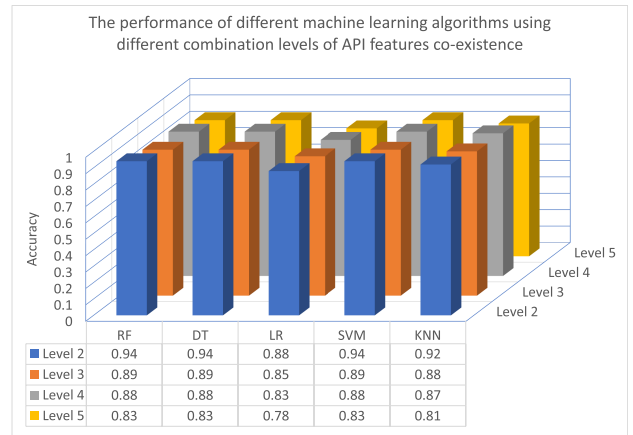


FIGURE 11. The performance of different machine learning algorithms using different combination levels of API features co-existence in Drebin dataset.

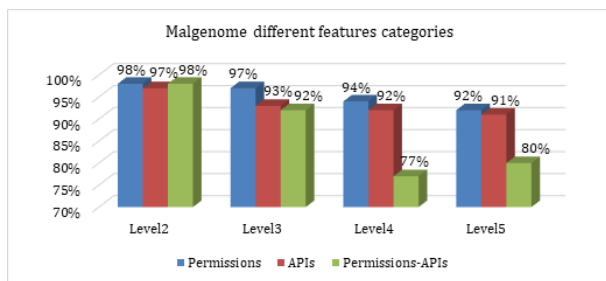


FIGURE 9. RF, DT and SVM accuracy for different feature categories at different levels for Malgenome dataset.

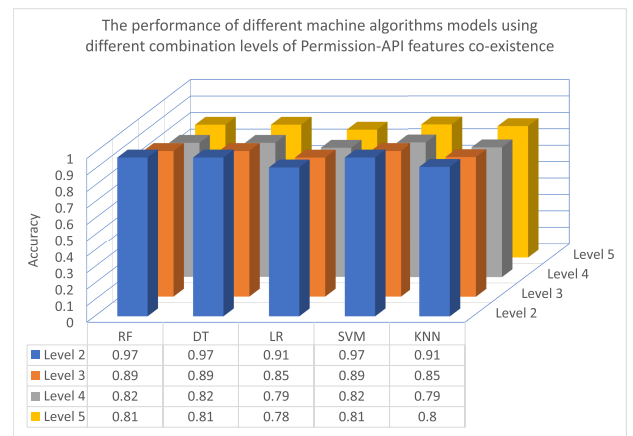


FIGURE 12. The performance of different machine learning algorithms using different combination levels of permission-API features co-existence in Drebin dataset.

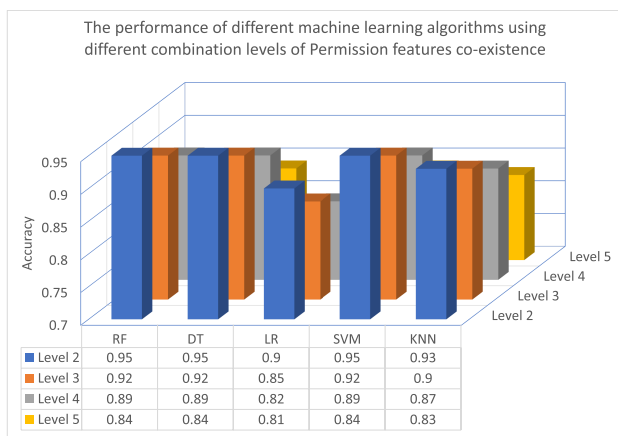


FIGURE 10. The performance of different machine learning algorithms using different combination levels of permissions features co-existence in Drebin dataset.

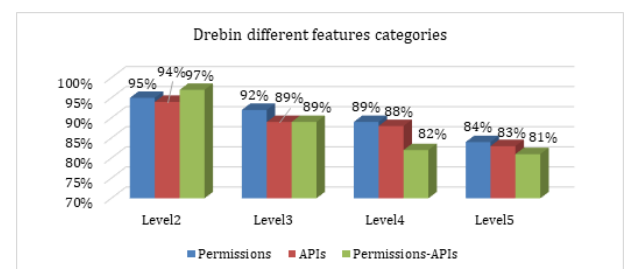


FIGURE 13. RF, DT and SVM accuracy for different feature categories at different levels for Drebin dataset.

did not consider the API frequencies. In this section, the paper considers the API features frequencies in CIC_MALDROID2020 to investigate its effectiveness in the co-existence-based detection. This work classified the API features in an APK into “existed (1)” or “not existed (0)” based on the API frequencies in the APK. For this purpose, a predefined threshold value is set. Several random thresholds were tested to evaluate their classification accuracy and to select an appropriate threshold. The best results were

achieved using the average of API calls frequencies as the threshold value. In this experiment we relied on the average of API frequencies as a threshold to convert API frequencies to API features existences. That is, if an API frequency in some APK is greater than or equal to the threshold value, this API is considered as “existed” (its value is set to 1) in the APK, otherwise, it is considered as “not existed” (its value is set to 0). After that, the API co-existence matrix was extracted, and a classification based on the co-existence of features was applied.

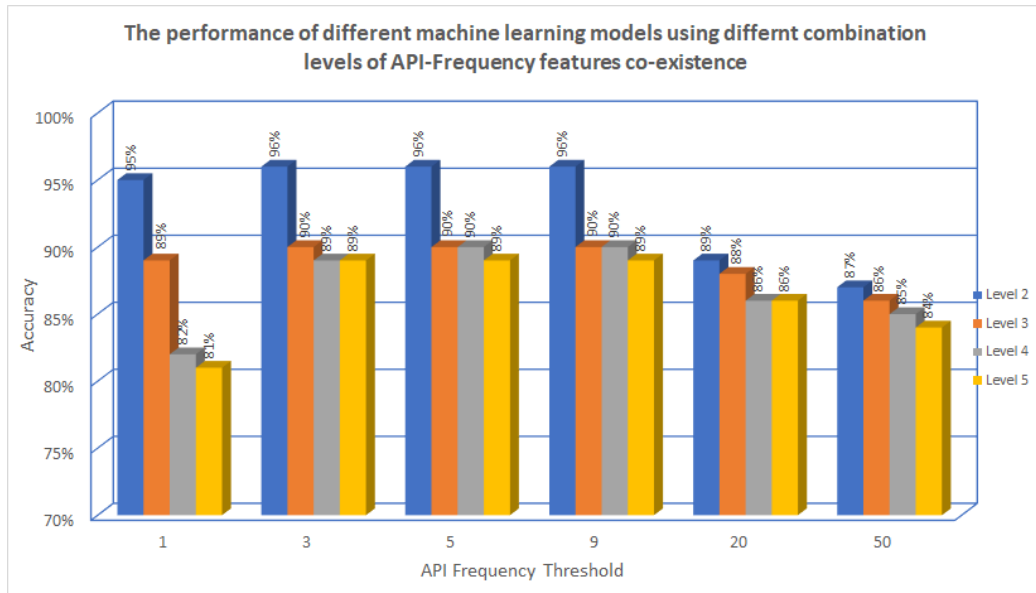


FIGURE 14. Accuracy of features co-existence using different API thresholds.

Figure 14 shows the results of the experiment using different threshold values, where the average of API frequencies is 9 in this experiment. The experiment was conducted using Random Forest algorithm and the CIC_MalDroid2020 dataset. Obviously, the accuracy of co-existence detection approach based on API frequencies varies at different threshold values. The results show that the accuracy based on the thresholds greater than the average of API frequencies is smaller than the accuracy achieved when using the average of API frequencies as a threshold. This is due to the fact that increasing the threshold value decreases the number of API features used in detection sharply, which negatively affect the classification process. Similarly, the accuracy does not change or decreases when decreasing the threshold values below the average of API frequencies. Decreasing the threshold value will achieve the same results when not using API frequencies. This proves that using the average of API frequency as a threshold value achieves the best results.

Figure 15 compares between the accuracy achieved by applying Random Forest algorithm on the co-existed API features dataset extracted from the CIC_MalDroid2020 dataset, and that achieved by applying Random Forest algorithm using co-existed frequent API features dataset (using API frequency) extracted from the same dataset. Obviously, using API frequency achieves better accuracy than using API existence only. This may refer to the assumption that frequent call of an API reflects the importance of this API to the APK. That is, this approach filters insignificant API features that may negatively affect the classification process.

F. A COMPARISON WITH THE STATE-OF-THE-ART

In this section, a comparative evaluation between the proposed approach and a state-of-the-art co-existence permission-based detection technique. Arora and Peddoju

in [44] proposed the Perm-Pair detection technique, which uses permission pairs to detect Android malware. The authors used three different malware datasets: Genome [17], Drebin [13], and Koodous [45]. The method depends on extracting permission pairs from each application and connecting those pairs using the graph structure for malware detection. The unique permissions for each app were represented by a vector V , and a pair of permissions was connected using a weighted edge. The weight of an edge increases with the existence of the same permission pair from different applications in the dataset. After processing all apps in the dataset, the edge weights were divided by the number of applications in the dataset to normalize weights across different dataset graphs. After generating separate Genome (GG), Drebin (GD), and Koodous (GK) graphs, they were merged into a single graph (GM) to represent the malicious graph. The final model consisted of two graphs: a malicious graph and a normal graph. In the detection phase, for every testing application, its permissions pairs were extracted, then two scores were calculated: a malicious score and a normal score, by searching every permission pair in GM and GN and adding the corresponding weights. Therefore, if the malicious score is greater than the normal score, the application was classified as malware and vice versa. The graph merging process of the three malware graphs included two types of edges: disjoint and common. All the disjoint edges were added to GM directly. The weighted sum method was used for determining a single weight of common edges because common edges have three different weights. Finally, the system finds the maximal set of irrelevant edges that do not affect the detection results and removes them.

The authors focused only on permission pairs because there were so many permission patterns when analyzing groups of more than two features. However, the proposed

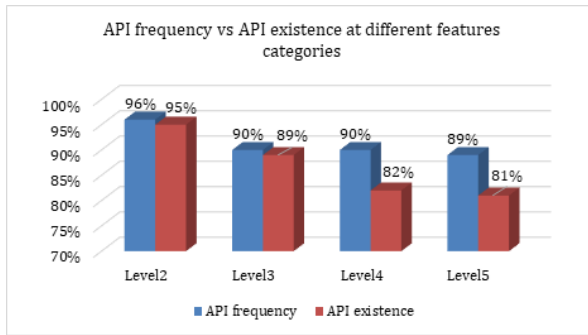


FIGURE 15. The accuracy of conducting RF algorithm using the API existence dataset and the API frequency dataset at different levels of co-existence.

TABLE 7. Drebin and malgenome datasets size in PermPair and the proposed approach.

Technique	Drebin	Genome	Benign
Perm-pair	2744	1264	5993
The proposed work	5550	1260	5550(used with Drebin) 1260(used with Malgenome)

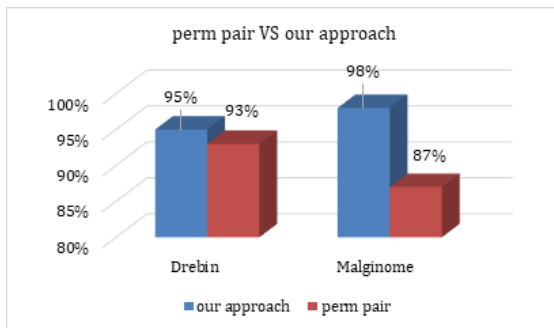


FIGURE 16. Accuracy comparisons between the proposed approach and the PermPair method.

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.

V. 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.

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-used-smartphone-market-will-grow-11-2-2024?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-os-users>
- [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/wp-content/uploads/2017/02/ICIT-Analysis-Signature-Based-Malware-Detection-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-for-android-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://kailinixtutorials.com/dex2jar-android-java/>
- [12] *Androzo*. Accessed: Jul. 30, 2022. [Online]. Available: <https://androzo.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://contagiomindump.blogspot.it/>
- [17] *Malgenome Project*. accessed: Jul. 30, 2022. [Online]. Available: <http://www.Malgenomeproject.org>
- [18] C.-F. Tsai, Y.-C. Lin, and C.-P. Chen, "A new fast algorithms for mining association rules in large databases," in *Proc. IEEE Int. Conf. Syst., Man Cybern.* San Francisco, CA, USA: Morgan Kaufmann, Oct. 1994, pp. 487–499.
- [19] A. Lab. (2017). *Amd Dataset*. Accessed: Oct. 30, 2022. [Online]. Available: <https://www.kaggle.com/datasets/blackarcher/malware-dataset>
- [20] V. Avdiienko, "Mining apps for abnormal usage of sensitive data," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, vol. 1, May 2015, pp. 426–436.
- [21] Y. Aafer, W. Du, and H. Yin, "Droidapiminer: Mining api-level features for robust malware detection in android," in *Security and Privacy in Communication Networks*, T. Zia, A. Zomaya, V. Varadharajan, and M. Mao, Eds. Cham, Switzerland: Springer, 2013, pp. 86–103.
- [22] V. M. Afonso, M. F. de Amorim, A. R. A. Grégio, G. B. Junquera, and P. L. De Geus, "Identifying Android malware using dynamically obtained features," *J. Comput. Virology Hacking Techn.*, vol. 11, no. 1, pp. 9–17, 2015.
- [23] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, and L. Cavallaro, "DroidScribe: Classifying Android malware based on runtime behavior," in *Proc. IEEE Secur. Privacy Workshops (SPW)*, May 2016, pp. 252–261.
- [24] H. Cai, N. Meng, B. G. Ryder, and D. Yao, "DroidCat: Effective Android malware detection and categorization via app-level profiling," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 6, pp. 1455–1470, Jun. 2019.
- [25] A. F. A. kadir, N. Stakhanova, and A. Ghorbani, "An empirical analysis of Android banking malware," Tech. Rep., Nov. 2016.
- [26] M. Sun, M. Zheng, J. C. S. Lui, and X. Jiang, "Design and implementation of an Android host-based intrusion prevention system," in *Proc. 30th Annu. Comput. Secur. Appl. Conf.* New York, NY, USA: Association for Computing Machinery, 2014, pp. 226–235, doi: [10.1145/2664243.2664245](https://doi.org/10.1145/2664243.2664245).
- [27] *Google Play Store*. Accessed: Jul. 30, 2022. [Online]. Available: <https://play.google.com/store/games>
- [28] D. Son. (2019). *Apktool—Tool For Reverse Engineering Android Apk Files*. Accessed: Oct. 30, 2022. [Online]. Available: <https://securityonline.info/apktool-reverse-engineering-android-apk-files/>
- [29] *Mcafee*. Accessed: Jul. 30, 2022. [Online]. Available: <https://www.mcafee.com/>
- [30] G. Baldini and D. Geneiatakis, "A performance evaluation on distance measures in KNN for mobile malware detection," in *Proc. 6th Int. Conf. Control, Decis. Inf. Technol. (CoDIT)*, Apr. 2019, pp. 193–198.
- [31] Y. Zhang, Y. Yang, and X. Wang, "A novel Android malware detection approach based on convolutional neural network," in *Proc. 2nd Int. Conf. Cryptogr., Secur. Privacy*, Mar. 2018, pp. 144–149.
- [32] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, and S. Anwar, "Static malware detection and attribution in Android byte-code through an end-to-end deep system," *Future Gener. Comput. Syst.*, vol. 102, pp. 112–126, Jan. 2020, doi: [10.1016/j.future.2019.07.070](https://doi.org/10.1016/j.future.2019.07.070).
- [33] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "MARVIN: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *Proc. COMPSAC*, vol. 2, Jul. 2015, pp. 422–433.
- [34] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: Effective and efficient behavior-based Android malware detection and prevention," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 1, pp. 83–97, Jan./Feb. 2018.
- [35] S. Chen, M. Xue, Z. Tang, L. Xu, and H. Zhu, "StormDroid: A streaming-glized machine learning-based system for detecting Android malware," in *Proc. 11th ACM Asia Conf. Comput. Commun. Secur.* York, NY, USA: Association for Computing Machinery, May 2016, pp. 377–388, doi: [10.1145/2897845.2897860](https://doi.org/10.1145/2897845.2897860).
- [36] (2020). *CIC_MALDROID2020 Dataset*, Canadian Institute for Cybersecurity. Accessed: Oct. 30, 2022. [Online]. Available: <https://www.umb.ca/cic/datasets/maldroid-2020.html>
- [37] S. Y. Yerima and S. Sezer, "DroidFusion: A novel multilevel classifier fusion approach for Android malware detection," *IEEE Trans. Cybern.*, vol. 49, no. 2, pp. 453–466, Feb. 2019.
- [38] H. L. Thanh, "Analysis of malware families on Android mobiles: Detection characteristics recognizable by ordinary phone users and how to fix it," *J. Inf. Secur.*, vol. 4, no. 4, pp. 213–224, 2013.
- [39] E. B. Karbab, M. Debbabi, A. Derhab, and D. Mouheb, "MalDozer: Automatic framework for Android malware detection using deep learning," *Digit. Invest.*, vol. 24, pp. 48–59, Mar. 2018.
- [40] Z. Aung and W. Zaw, "Permission-based Android malware detection," *Int. J. Sci. Technol. Res.*, vol. 2, no. 3, pp. 228–234, 2013.
- [41] P. Branco, L. Torgo, and R. P. Ribeiro, "A survey of predictive modeling on imbalanced domains," *ACM Comput. Surv.*, vol. 49, no. 2, pp. 1–50, Aug. 2016.
- [42] M. Kavitha and S. T. Selvi, "Comparative study on Apriori algorithm and Fp growth algorithm with pros and cons," *Int. J. Comput. Sci. Trends Technol.*, vol. 4, 2016.
- [43] D. Gong. (2022). *Top 6 Machine Learning Algorithms for Classification*. Accessed: Dec. 4, 2022. [Online]. Available: <https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501>
- [44] A. Arora, S. K. Peddoju, and M. Conti, "PermPair: Android malware detection using permission pairs," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 1968–1982, 2020.
- [45] *Koodous: Collective Intelligence Against Android Malware*. Accessed: Jul. 30, 2022. [Online]. Available: <https://koodous.com/>



ESRAA ODAT received the B.Sc. degree in computer information systems and the M.Sc. degree in data science from the Jordan University of Science and Technology, Jordan in 2020 and 2022, respectively. Her research interests include machine learning, deep learning, and cybersecurity.



QUSSAI M. YASEEN received the B.Sc. degree in computer science from Yarmouk University, in 2002, the M.Sc. degree in computer science from the Jordan University of Science and Technology, in 2006, and the Ph.D. degree in computer science from the University of Arkansas at Fayetteville, AR, USA, in 2012. His research interests include malware analysis, insider threat, and computer networks security. He has published many papers in network security, insider threat, the IoT security, machine learning with cybersecurity (security analytics), and spam filtering. Moreover, he has served as a chair/TPC member/reviewer for many events, conferences, and journals in this field and other information technology fields.

• • •