Permission Extraction Framework for Android Malware Detection

Ali Ghasempour[1], Nor Fazlida Mohd Sani[2], Ovye John Abari[3]
Department of Computer Science, Universiti Putra Malaysia
43400 UPM Serdang, Selangor, Malaysia

***Abstract*—Nowadays, Android-based devices are more utilized than other Operating Systems based devices. Statistics show that the market share for android on mobile devices in March 2018 is 84.8 percent as compared with only 15.1 percent iOS. These numbers indicate that most of the attacks are subjected to Android devices. In addition, most people are keeping their confidential information on their mobile phones, and hence there is a need to secure this operating system against harmful attacks. Detecting malicious applications in the Android market is becoming a very complex procedure. This is because as the attacks are increasing, the complexity of feature selection and classification techniques are growing. There are a lot of solutions on how to detect malicious applications on the Android platform but these solutions are inefficient to handle the features extraction and classification due to many false alarms. In this work, the researchers proposed a multi-level permission extraction framework for malware detection in an Android device. The framework uses a permission extraction approach to label malicious applications by analyzing permissions and it is capable of handling a large number of applications while keeping the performance metrics optimized. A static analysis method was employed in this work. Support Vector Machine (SVM) and Decision Tree Algorithm was used for the classification. The results show that while increasing input data, the model tries to keep detection accuracy at an acceptable level.**

Keywords—Malware detection; android device; operating system; malicious application; machine learning

I. Introduction

In recent time, Android is the most famous platform for mobile devices [1]. Saving and storing confidential data such as banking information or contact numbers is part of every mobile device. Hence providing a barrier between these information and an attacker is highly needed. Nowadays antiviruses are greatly developed and provide a wide range of options based on user needs. However, studies show most Android users do not rely on antiviruses to secure their phones [2]. So, this can be a great motivation for the attacker to focus on the Android platform. Kaspersky, which is one of the biggest security solutions company released information related to malicious activity on the Android platform in 2018. Based on the statistics, 5,730,916 malicious packages were detected by their lab [3]. Thus, protecting Android devices from misuse or any malicious application is important and needed.

Mobile malware can be classified into three (3) groups [4]. These are Malware, Grayware, and Spyware. The Malware focuses on gaining access to personal data or damaging to the hardware (mobile device). One of the main aims of Malware attacks is unprivileged access to user personal information. Malware attacks are SMS, Bluetooth, GPS, and Root attacks. SMS attack is mostly related to phishing and adware. In Bluetooth attack, an attacker can steal user personal information or location services. GPS attacks can compromise GPS devices on mobile phones to steal user location. In Root attack, an attacker can get privileged access to the phone operating system to install or remove applications. Grayware is an attack that mostly focuses on the marketing side without any damaging to phone or user data. Spyware is the most frequent type of attack related to the mobile phone which steals user data and sends it to an unwanted application rather than the original one. Android malware can further be specifically categorized into Worm, Trojan, Back-doors, Botnet, Spyware and Ransomware [5].

To analyze the behavior of application in the Android platform, two methods of analysis are commonly used. These are Dynamic and Static analysis [5]. Behavioral or dynamic focus on application behavior during run time. For example, requested permissions while an application is running can determine application intend. Additionally, a system call is one of the main features which can be measure for application behavior in runtime. To address why system call can be named as dynamic analysis, an Android operating system is based on Linux. A system call is provided by the application to request specific resources whether hardware or library from Linux kernel. Requesting irrelevant resources for an application can be marked as suspicious activity. [6].

The static analysis is mostly applicable for analyzing application by disassemble package and extracting source. Like dynamic analysis, different approaches exist in static analysis. Some researches focused on finding a sequence of op-code that reflects the malicious activity. The reason why op-code can be named as a feature is that every instruction in the computer program follows a specific structure. In this structure op-code determine the action of this instruction and the rest is an address in memory. Malicious packages may follow the same action in instruction, and so, op-code can be a good classifier for detection.

Permission is a famous feature for Android malware detection and it expose the exact intent of the application. It is designed to protect user privacy on the Android platform. Permission is requested by an application to authorize itself to access specific hardware or software resource. For instance, the sensitive user data (such as SMS or contacts) or system features (such as a camera or internet). Based on application nature, the system may grant permission or ask the user to grant it. Basically, no application has the authorization to access or read other application data, system files, and user private data (such as SMS). Details related to permission are discussed in the next section. Fig. 1 illustrates how Android malware can be categorized based on different methods of analysis.

In Malware detection, after the feature selection phase, designing the model using real-world input sample data is the next phase. In the design phase, different statistical and mathematical techniques are used to prune unnecessary features from input data as well as to highlight selected features. A well-structured model can improve detection accuracy. For the classification phase, existing works mostly used machine learning, deep learning, and statistical methods to carried out the classification. Although most of the techniques are based on mathematics but the terms are different.

Machine learning is one of the basis and famous techniques used in the classification phase. It is a general term and contains many different algorithms inside. The main idea is to design a model based on known input data to predict unseen input data. In Machine learning, we try to predict as precious as possible for unknown input data which is called test data. Most of the algorithm follows two steps; train and test. In the training phase, the machine tries to make a relevant model based on labelled input data and whereas, in the test phase, the machine will predict based on its knowledge of previous learning. Moreover, machines may work on an unforeseen situation where the train phase does not exist. Therefore, machines try to find a pattern to distinguish between objects. Two main machine learning techniques are classification (Supervised learning) and clustering (Unsupervised learning) [7]. Supervised learning uses labelled data for the classification. Here, the features need to be defined by the operator, and results are described through the mapping of input to output by rules that are provided by the supervisor. Unsupervised learning focuses on finding a pattern in unlabelled or not fully labelled data. The results mostly are the grouped of input data. In clustering problems, machines try to group data with the same features instead of classifying them.

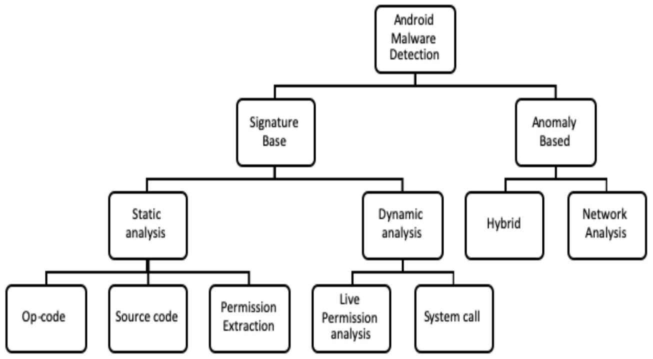

Fig. 1. The different Methods in Android Malware Detection.

Recently, deep learning has been used for classification. In deep learning, instead of focusing on linear analysis, we target multiple levels of abstraction. Data representation in each layer transfers to the next level to achieve a more accurate level of abstraction for input data. Also in this method, feature selection is used to improve the accuracy [8]. However, one of the drawbacks of the deep learning method is that it uses a lot of resources and takes a long time due to the high level of decomposition to complete the classification. The statistical method is used for the classification and to differentiate between applications. The statistical model uses less computational resources than machine learning techniques [9].

In this work, the authors focused on permission as an input feature. Because we have only one feature to analyse, the model is designed based on binary classification. The proposed model used different statistical techniques to boost computation while keeping accuracy high. Permission Support and Permission Ranking were used in the early steps of this model. Principal Component Analysis (PCA) was used to find different views of permission correlation. Dimensionally reduction using Kaiser and Cumulative techniques was implemented to select the most significant permissions. Selected permissions were compared to the results of Li [10] to select similar permission. Also, separate analysis has been done on our method without any overlap with the existing results. The last phase of this work focused on classification where Supervised Machine Learning approaches (Support Vector Machine and Tree-based algorithm) were used for the classification because of their popularity. In summary, the main contributions of this paper are as follows:

Proposed a multi-level permission extraction framework that improved malware detection accuracy in Android devices.

The framework designed was able to handle a large number of applications while keeping the performance metrics optimized.

The organization of the paper is as follows. Section 2 presents the literature review or related work. Section 3 gives the details about the methodology used. Section 4 gives the details about the proposed framework. Section 5 is the experiments setting and results. Section 6 gives a summary of the paper.

II. Literature Review

The important concepts in the understanding of malware detection in Android devices were discussed. The current trend and techniques used to develop an Android based malware detection model are also explained in this section.

A. *Static based Analysis*

The static based analysis includes the following methods**.**

- Source Code

[11] proposed a model to extract the source code of application using machine learning and clustering. Firstly, the authors extract the source code and then tokenized it with the n-gram method and used a bag of words (BoW) algorithm,

which is one of the NLP algorithms. Then they used both supervised and unsupervised machine learning techniques to get their result. The authors used ensemble learning by repeating the experiment 10 times combining with other methods and use the majority of results to make predictions. With the SVM method, they achieved a 95% detection rate. However, with the clustering method, they obtained only a 72% detection rate. Although the ensemble method showed better result with 95.6% detection rate [11].

- Op-Code

[12] developed a Deep Android Malware Detection to get a sequence of op-code and used a convolutional neural network (CNN) in detecting malicious applications. They used a rich dataset for the training of the model. The authors used a one-layer convolutional neural network over three types of datasets. For a small dataset, they achieved a 97% accuracy and for the large and extra-large dataset, they got 78% and 86% accuracy respectively. Their results show that the model produces better results as the dataset and training time are increasing [12].

- Permission

[11] proposed a model for Android malware detection using machine learning. The authors used the ensemble technique to get more accurate results. The results for classification with the SVM algorithm were 87.9% and using ensemble learning with C.45, Random Forest, Random tree, SVM, Logistic regression was 89.4%. Using the clustering method, they achieved a 64.6% detection rate [11].

- Hardware and Permission

Multimodal learning is one of the neural network techniques that use multi-input as network input. [13] applied static analysis to extract features from android packages. The features the authors considered in their work are permission and hardware features. They two features were chosen because the requested permission can define access to sensitive data, and the hardware permission can access physical resources in mobile devices such as cameras. The authors used these features separately to achieve a higher degree of accuracy. They further combined both features to improve and obtain a better performance result. Grid search was used over selected hyperparameters to optimize performance value. The analysis demonstrates that the overall accuracy achieved was 94.5%.

- API Tag and Permission

CENDroid was developed by [14] as a static-based Android malware detection based on API tags and permission. They formed a different combination of features for analysis behaviour and set up five machine learning algorithms for the classification. Afterward, three ensemble combination techniques were used. CENDroid was based on the clustering training dataset into an optimized number of clusters and then used ensemble learning for each of them. Ensemble model help to improve the performance as it can handle dissimilarity due to misclassification. The proposed model was trained and tested on different datasets and results revealed that linear SVM achieved a higher degree of 96.13% accuracy as compared to other classifiers. In ensemble learning, the weighted majority voting method hits the highest accuracy of 97.38% as compared to the staking and majority voting [14].

B. *Dynamic based Analysis*

Th system call method in a dynamic based analysis is discussed below.

- System Call

M0droid was proposed by [15] as a client/server architecture for detecting the malicious application using system call analysis. In this model, when an application is installed on a phone, the client check hash of APK file with his database to detect whether it is listed or not. If it is not listed, the server will send the APK file to do further investigation. The application will be run on an isolated environment and all of the system calls will be captured. After that, a vector that was normalized using z-score store how many times the system call is repeated. A signature is made from application and Spearman’s rank correlation was used to compare the database and labelled the application. After setting the threshold value to 0.92, the result obtained was 60%.

C. *Hybrid Analysis*

[16] used a hybrid approach (a combination of analyzed source code and getting the requested permission) to proposed their model for malware detection. The developed model is capable of extracting three items. First, meta-data of application which contain permission of applications. Second, the binary of *Dex* code and library used in APK. Third, analysing the *Dalivk* assembly. A fuzzy fingerprint was used to make DNA for each application. Further, the authors proposed another malware detection, called ROAR for comparing the mentioned feature from the new application. Their framework is divided into three parts, family fingerprinting, peer fingerprinting, and merged fingerprinting. They achieved an accuracy rate of 95% on the last two methods and 85% for family fingerprinting.

[17] proposed a model by extracting nine (9) features from every application. The *DroidBox* is used to emulate the application in a virtual environment. Cryptography material, network operation, file operation, *Dexclass* load, information leak, sent SMS, phone call, service start, receive reaction and system call is used for malware detection. A Chi-Square algorithm was applied for dimension reduction and further used ensemble learning and combined different machine learning techniques. The best success rate among the different datasets and machine learning methods was 96.42%.

[18] discussed the importance of Android devices on the current Internet of Things (IoT) space. It was revealed that securing IoT devices that are running on Android is an issue. The authors used Factor Analysis of Information (FAIR) model to measure the risk associated with the IoT devices. In a situation of threat, they considered Situational Awareness (SA) to recognize the environmental elements. The proposed model contains three layers. The first layer, using machine learning, detect malicious behaviour of the attacker. The second layer mapped the selected feature in the first layer to

LEF factor of the FAIR model. The third layer, which is the decision-making layer finalized the process. The results demonstrate that with different machine learning techniques, different degrees of accuracy can be achieved. Based on the provided data, linear SVM has the highest accuracy of 99% as compared to other methods.

Author in [19] focused on different features to differentiate between Android applications and proposed a malware detection model. Op-code and API calls are choosing as desire features for analysing. The proposed model contains five layers. After feature extraction from APK files, Op-codes are tokenized by n-gram technique and API call by API frequency vector. A Convolutional neural network (CNN) was used as the main analyser in this model. Due to the high number of dimensions, principal component analysis (PCA) was used to distinguish significant features. The Neural network outputs are passed to feature fusion to achieve a higher degree of accuracy. SoftMax was used as the classification algorithm. The results focus on achieving accuracy while keeping runtime low. On 0.034 seconds, 95.1% accuracy achieved.

D. Significant Permission Identification

To achieve malware detection with high certainty, the Significant Permission Identification is used to achieve a high detection rate with the lowest possible permissions for analysing. This method focuses on permission which has a high chance to labelled as malware and omit permissions with low effect on our detection. To implement, three-layer data filtering is used for analysing real-time data. These are: (1) permission ranking with negative rate, (2) support based permission ranking and (3) permission mining with association rules. After applying these permissions, a supervised machine learning model is trained to detect further application.

- Data Pruning

The first step for analysing is to eliminate the necessity of considering all of the permissions for analysing. Due to the high number of applications in our dataset and each application consist of many permissions, it takes time to analyse all of them. We use Multi-Level Data Pruning (MLDP) to detect the most affectable permissions. The following are discussions about MLDP solution.

1) Permission Ranking with Negative Rate (PRNR): In every application, any requested permission reflects the need of the application. As far as we know the malicious application follows a specific subset of permission, then there will be no need to study all of the permissions. The algorithm only focuses on dangerous permissions and frequently requested permissions. Also, it can separate malicious applications from the benign applications with rare permissions requested by malware applications. For increasing the detection rate, the algorithm is omitting the same permission requested by both classes. To come up with one solution, permission ranking with negative rate is used to distinguish not only high-risk permission but also the permission used by benign applications. The goal is to differentiate normal and malware attacks.

We define two matrices, M and B. M is defined as the list of permission used by malicious application and B is the list of permission used by the benign application. M_{ij} says whether permission jth is used by the ith malicious application or not. If the answer is yes then 1 otherwise 0. The same goes for B_{ij} for benign applications.

In the first step, we need a balance between malicious applications and benign applications. In this work, the size of the benign application is 310926 while malicious is only 5494 applications, which caused skew in our model. Therefore, it is used in the equation 1 to find the support of each permission in a larger dataset and then scale down that permission to match smaller datasets. In this case number of benign B is more than malicious M.

$$S_B\left(P_J\right) = \frac{\sum_i B_{ij}}{Size(B_J)} * Size\left(M_J\right) \tag{1}$$

P_j describes jth permission and $S_B(P_j)$ is the support of jth permission in the B matrix. After rescaling, we can deploy PRNR in the equation 2:

$$R\left(P_j\right) = \frac{\sum_i M_{ij} - S_B(P_j)}{\sum_i M_{ij} + S_B(P_j)} \tag{2}$$

The $R(P_j)$ shows the rate of jth permission. The range for an answer could be from [-1, 1] which 1 shows the permission P_j exists in the malicious dataset, so it can be categorized as danger permission. Also, if it is -1, it means that permission exists in the benign dataset which is low-risk permission. If 0, it means that it doesn't have a special impact on the detection. After that, two lists are generated from $R(P_j)$ in a form of ascending or descending order. To continue, the Permission Incremental System (PIS) is proposed for getting top permission from the benign and malicious list with the following metrics: True Positive, False Positive, Precision, recall, accuracy and F-measure. The aim is to find out those permissions which have much effects on the whole of the datasets. For the 310926 benign applications and 5494 malicious applications, all of the requested permissions are 135 and decreased to 95 with PRNR.

2) Support Based Permission Ranking (SPR): To narrow the result of PRNR and support each permission is considered that we focus on how many times one permission is repeated in the whole of the dataset or if permission exists only on a specific dataset. Therefore, we can omit permission with the lowest support. To implement this, the Permission Incremental System (PIS) is used to find the most supported permission. Our results show that 25 permission out of all the permissions is supportive.

3) Permission Mining with Association Rule (PMAR): After two steps pruning data, we get 25 most significant permission. To look closer to these permissions, it seems that some of the permissions are formatted as pairs. As an example, permission WRITE_SMS always comes with READ_SMS and so, we can consider them as one. To find this association between permissions, we used association rules. These rules are used for discovering relations between entities in the dataset. It only focused on high confidence permissions. The algorithm which is used for detecting permissions is Apriori [20], which is one of the famous algorithms in association rules. With 96.5% confidence and 10% minimum support, 3 permission were found which can be removed due to its relative to another permission. Finally, 22 permissions were left as the most important permissions as shown in Table I.

- Machine Learning on Significant Permission

For the detection, a supervised machine learning method was used. Support Vector Machine (SVM) was applied to small data to test the Multi-Level Data Pruning model. SVM needs two classes from the training dataset to differentiate its hyperplane. One class is map to the malicious and the other class to the benign while the input data would be to unknown application. The application is then mapped to vector space to decide whether it is benign or malicious.

To check the applicability of MLDP, 67 machine learning techniques were used. The results of the learning algorithm over the original dataset were compared and applied algorithms on the MLDP dataset. Also, tree base techniques like decision tree have better result but due to high features among the data, this method consumed a lot of memory and thus greatly reduce the accuracy of the results.

Functional tree (FT) and Random Forest are more likely to have better Recall than the other algorithms. However, Random Forest consumes more memory and time to analyse data. One of the main issues with this algorithm is that while the number of datasets increasing, FPR and other performance metrics are slightly decreasing. Table II shows that as we moved from 2650 to 54694 applications, the false positive rate increases to 4.85%, and this is really high.

E. Comparison between Different Techniques

In this section, we conclude that each method has its own advantages and disadvantages. These pros and cons can be shown in terms of the accuracy of detection or overhead on computational resources. Therefore, there is no specific best method for investigation in the area of Android. Fig. 2 illustrates the comparison between different techniques in terms of accuracy.

TABLE I. MOST SIGNIFICANT PERMISSION BY MLDP

MLDP	
22 Permissions	
ACCESS_WIFI_STAT	READ_LOGS
CAMERA	READ_PHONE_STATE
CHANGE_NETWORK_STATE	READ_SMS
CHANGE_WIFE_STATE	RECEIVE_BOOT_COMPLETE
DISABLE_KEYGUARD	RESTART_PACKAGES
GET_TASKS	SEND_SMS
INSTALL_PACKAGES	SET_WALLPAPER
READ_CALL_LOG	SYSTEM_ALRERT_WINDOW
READ_CONTACTS	WRITE_APN_SETTING
READ_EXTRERNAL_STORAGE	WRITE_CONTACTS
READ_HISTORY_BOOKMARKS	WRITE_SETTINGS

TABLE II. RESULTS FOR DIFFERENT DATASETS

Num_of_Malapp	2650	5494	54694
Precision	98.83%	97.54%	95.15%
Recall	94.4%	93.62%	92.17%
FPR	1.17%	2.36%	4.85%
FM	94.97%	95.54%	93.63%
ACC	96.47%	95.63%	93.67%

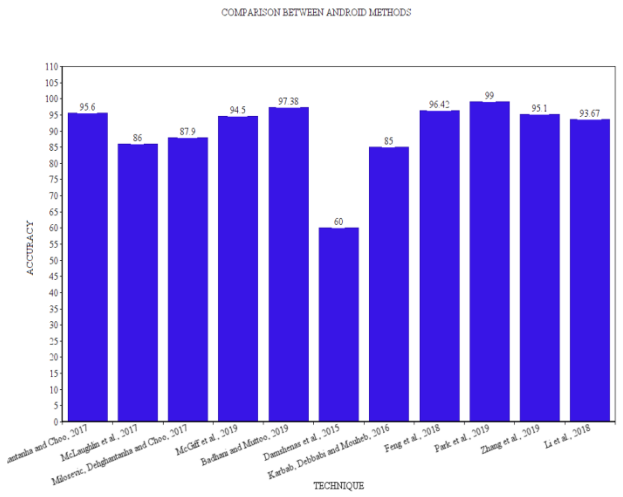

Fig. 2. Comparison between the different Methods in Android Malware Detection.

III. METHODOLOGY

Here, we discussed the simulation environments, data collection, and research methodology.

A. Simulation Environment

In this work, Python 3.6.1 is used as the base scripting language. Python provides a wide range of libraries and functions. Additionally, there are a lot of Android packages analysis tools that are written in Python. Operating systems vary during different phases of analysis. Windows 10 is our base operating system in this analysis but however in some cases, Linux CentOS 6.8 is used.

The data extraction platform needs strong computational power for the early stage of the study. Because of the high number of input data set, it is good to use a high-frequency CPU to extract specific data from APK files. In our work, we used two different platforms. First, to only inspect a small amount of test data in making the machine learning model, we run it on Intel i7-7500U, 2.70 GHz, 16 GB RAM. Second, most of the computation is conducted on Intel Xenon L5640 2.70 GHz, 10 GB RAM.

B. Datasets Collection

The scope of Android malware detection only focuses on real applications. These applications can be collected from different common markets like Google play store. Over the years, researchers collected some labelled datasets to ease their researches. Some of these datasets are kept confidential with related to specific projects and accessing them requires special permission. But some of these datasets are accessible for researchers.

In this research, M0droid project dataset [21] with overall 410 labelled applications is used. This dataset contains 200 malware applications and 210 normal applications. The other dataset used was collected from different projects in the University of Brunswick [22]. This dataset is highly categorized and labelled. Number of applications in this dataset is over 5000 and Android packages with 2000 normal applications and 3000 malware applications.

Also, the AndroZoo dataset [23] has granted permission to this research to use their dataset. This dataset is made up of about 8,000,000 applications. Android PRAGuard Dataset [24] is another dataset that was used with almost 10479 applications. Most of the applications are labelled. Android Malware Dataset (AMD) [6], is one of the largest malware datasets in Android. It contains 24553 applications that are grouped into 135 different groups. Up to 71 malware families have been seen in this dataset.

C. Android Packaging File (APK)

Application in the Android platform is delivered by APK file format. The focus of this study is on the APK file. Fig. 3 describes the architecture of the APK file. The *META-INF* includes manifest file, lib contain compiled code for any processor the device is using, res contains the non-compiled file, *assests* include application asset, *AndroidManifest.xm*l is used for the identification of applications and *classes.dex* contains the source code in *dex* format [25]. *Dex* files are only usable by *Dalvik* or *ART* compiler on Android devices. The APK file is compressed with *ZIP* format so we can get the desired file from unzipping the package. Different tools can be used to extract desire information.

D. Android Permission Structure

The proposed framework stands on Android permission. Permission is designed to add another level of security to protect user sensitive data such as SMS or contacts. In Android security architecture, no application by default has access to any action related to the operating system, other applications, or user private files. Permissions with low risk are automatically granted to the application. If application request high risk permission such as CAMERA, user acceptance is required to grant permission.

Dangerous permission can be asked in two methods:

1) Install time request: the system asks to grant all requested permission.

2) Runtime request: whenever services needed by an application, request permission from the user.

Permissions are highlighted by '*uses-permission*' in the Android manifest file. Some permissions are related to specific hardware feature and therefore, developer needs to mention '*uses-feature*' in the manifest file. This option helps the application runs on devices without the requested hardware. Google defined three protection levels for permission: *normal*, *signature,* and *dangerous.* These are discussed below.

- Normal Permission

Normal permission is a permission that is requested by application for accessing data or resources out of its box. Mostly, these permissions are granted by the system when the application is installed. According to Google on Android 9, the following as shown in Table III are normal permissions:

- Signature Permission

The system grants these permissions but the permissions need to be signed by the same application that requests permission. Table IV contains the list of signature permissions.

- Dangerous Permission

Dangerous permission is the type of permission that is requested to access user's private data or system files. When permission is declared as dangerous, user approval is necessary for allocating requested resources or data. Table V shows the dangerous permissions.

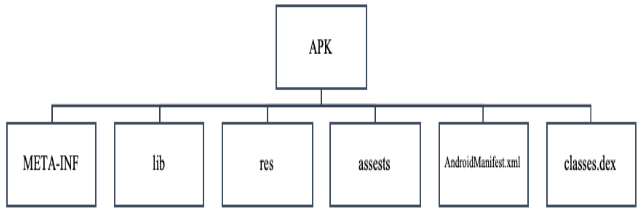

Fig. 3. APK File Structure.

TABLE III. NORMAL PERMISSION

ACCESS_LOCATION_EXTRA_COMMANDS	ACCESS_NETWORK_STATE	ACCESS_NOTIFICATION_POLICY	ACCESS_WIFI_STATE	BLUETOOTH	BLUETOOTH_ADMIN
BROADCAST_STICKY	CHANGE_NETWORK_STATE	CHANGE_WIFI_STATE	CHANGE_WIFI_MULTICAST_STATE	DISABLE_KEYGUARD	EXPAND_STATUS_BAR
FOREGROUND_SERVICE	GET_PACKAGE_SIZE	INSTALL_SHORTCUT	INTERNET	KILL_BACKGROUND_PROCESSES	MANAGE_OWN_CALLS
MODIFY_AUDIO_SETTINGS	NFC	READ_SYNC_SETTINGS	READ_SYNC_STATS	RECEIVE_BOOT_COMPLETED	REORDER_TASKS
REQUEST_COMPANION_RUN_IN_BACKGROUND	REQUEST_COMPANION_USE_DATA_IN_BACKGROUND	REQUEST_DELETE_PACKAGES	REQUEST_IGNORE_BATTERY_OPTIMIZATIONS	SET_ALARM	SET_WALLPAPER
SET_WALLPAPER_HINTS	TRANSMIT_IR	USE_FINGERPRINT	VIBRATE	WAKE_LOCK	WRITE_SYNC_SETTINGS

TABLE IV. SIGNATURE PERMISSION

BIND_ACCESSIBILITY_SERVICE	BIND_AUTOFILL_SERVICE	BIND_CARRIER_SERVICES	BIND_CHOOSER_TARGET_SERVICE	BIND_CONDITION_PROVIDER_SERVICE
BIND_DEVICE_ADMIN	BIND_DREAM_SERVICE	BIND_INCALL_SERVICE	BIND_INPUT_METHOD	BIND_MIDI_DEVICE_SERVICE
BIND_NFC_SERVICE	BIND_NOTIFICATION_LISTENER_SERVICE	BIND_SCREENING_SERVICE	BIND_TELECOM_CONNECTION_SERVICE	BIND_TEXT_SERVICE
BIND_TV_INPUT	BIND_VISUAL_VOICEMAIL_SERVICE	BIND_VOICE_INTERACTION	BIND_VPN_SERVICE	BIND_VR_LISTENER_SERVICE
BIND_WALLPAPER	CLEAR_APP_CACHE	MANAGE_DOCUMENTS	READ_VOICEMAIL	REQUEST_INSTALL_PACKAGES
SYSTEM_ALERT_WINDOW	WRITE_SETTINGS	WRITE_VOICEMAIL	*	*

TABLE V. DANGEROUS PERMISSION

CALENDAR	READ_CALENDAR	LOCATION	ACCESS_FINE_LOCATION	SMS	SEND_SMS
	WRITE_CALENDAR		ACCESS_COARSE_LOCATION		RECEIVE_SMS
CALL_LOG	READ_CALL_LOG	MICROPHONE	RECORD_AUDIO		READ_SMS
	WRITE_CALL_LOG	PHONE	READ_PHONE_STATE		RECEIVE_WAP_PUSH
	PROCESS_OUTGOING_CALLS		READ_PHONE_NUMBERS	STORAGE	READ_EXTERNAL_STORAGE
CAMERA	CAMERA		CALL_PHONE		WRITE_EXTERNAL_STORAGE
CONTACTS	READ_CONTACTS		ANSWER_PHONE_CALLS	*	*
	WRITE_CONTACTS		ADD_VOICEMAIL	*	*
	GET_ACCOUNTS		USE_SIP	*	*
	RECEIVE_MMS	SENSORS	BODY_SENSORS	*	*

IV. PROPOSED FRAMEWORK

The proposed framework contains different layers for data processing. It is called a multi-level permission extraction framework. The multi-level permission extraction framework contains methods from the previous researches as well as a novel method to improve the accuracy of the detection model. The general overview of the proposed framework is shown in Fig. 4.

A. *Permission Extraction from APK Files*

The first step in our framework is to extract permissions from APK files. As described previously, permissions exist in the Android manifest file. To read and extract desire information, specific tools are required which is discussed in the experimental section. After extracting permissions from both malware and benign datasets, data are mapped into a matrix. This matrix is called a feature matrix.

B. *Feature Matrix*

The feature matrix represents the existence of permission in the applications. In the feature matrix, each application's permissions are defined by Boolean variables. We defined two matrices, M and B. Matrix M is the list of permission used by the malicious application and matrix B is the list of permission used by the benign application. M_{ij} says whether permission j^{th} is used by the i^{th} malicious application or not. If the answer is yes then 1 otherwise 0. The same goes to B_{ij} for benign applications. Table VI is the sample matrix from the malware dataset.

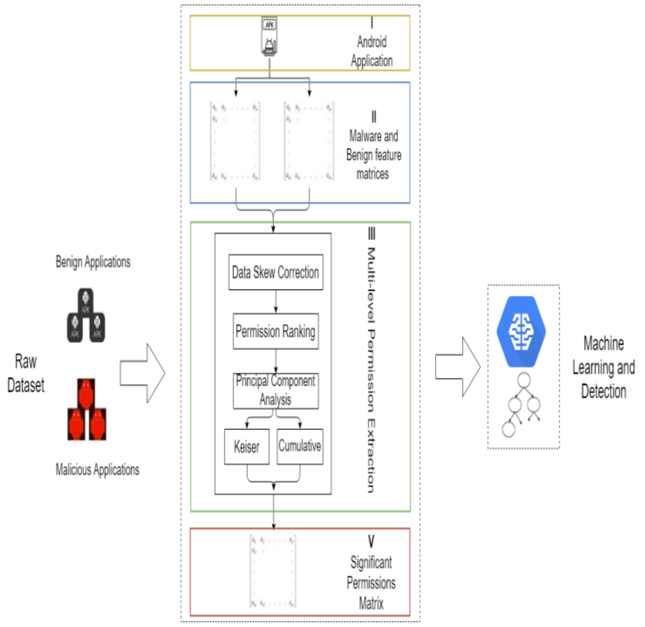
I
Android Application
II
Malware and Benign feature matrices
III
Multi-level Permission Extraction
Data Skew Correction
Permission Ranking
Principal Component Analysis
Keiser
Cumulative
V
Significant Permissions Matrix
Benign Applications
Raw Dataset
Malicious Applications
Machine Learning and Detection

Fig. 4. Proposed Multi-Level Permission Extraction Framework.

C. *Multi-Level Permission Extraction*

Multi-level permission extraction level is the proposed framework. In this framework, different levels of permission pruning are applied to achieve the most significant permissions. The proposed framework contains four main phases; data skew correction, permission ranking, principal component analysis, and two different statistical algorithms for identifying the most valuable permissions in the dataset. This framework hired some techniques from one of the states of arts research in Android malware detection [10].

- Data Skew Correction

In data analysis research, it is necessary that data in every category balanced to avoid any wrong or incorrect results. However, providing the same amount of data for every category is not possible. In this research, we used different sources for our datasets. The malware dataset size is around 25000 APK files as compare to 15000 APK files for the normal datasets. Also, we have around 8,000,000 APK files in both malware and benign dataset. This unbalances in malware and normal data category may lead to inaccurate results in the final steps.

In data skew correction layer, we used equation 3 to find support for each permission in the larger datasets and then scale down that permission to match smaller dataset. In this case, the number of malicious applications is more than the benign applications.

$$S_M(P_j) = \frac{\sum_i M_{ij}}{size(M_j)} \times size(B_j) \quad (3)$$

Where P_j describes j^{th} permission and $S_M(P_j)$ is the support of j^{th} permission in the malware matrix.

- Permission Ranking

In this step, permission ranking scheme has been used to hold the most frequent permissions in the dataset. Some permissions only happened in the context a few times, and hence these permissions have less support as compare to the common permissions. This scheme provides a more accurate view on what each permission represents in the datasets.

Term-Frequency Inverse Document Frequency (TF-IDF) technique is used to indicate the importance of each permission in all of the datasets. Term weighting methods are commonly grouped into supervised and unsupervised methods. The unsupervised or traditional term weighting methods are originated from the information retrieval field. The supervised weighting methods used the prior information of the training documents in predefined categories. The weight of a feature f with respect to a class represent the discriminating ability of f towards normal and attack classes. The higher the weight, the stronger the discriminating power of this feature in identifying the anomaly instances. Because *permission-based Vector* is by nature a bag of permission in the form of a vector, the *tf-idf* weighting method, where, $tf - idf$ is the weight of the feature f_i with regards to permission s_j is the product of $tf_{i,j}$ and $idf_{i,j}$ [26] and this is show mathematically in equation 4.

$$w_{f_i, s_j} = tf_{i,j} . idf_{i,j} \quad (4)$$

The matrices from the previous layer are passed into this layer and the value of all the rows is summed in one row. The new row represents how many times one permission is requested in a specific dataset. Then, the TF-IDF is applied to determine the impact of permission in the dataset. The weighting scheme result will omit the less frequent permissions and show the significant permissions in both datasets.

- Principal Component Analysis

Data transformation is required to come up with an efficient method to analyse a large amount of data. This method helps to reduce the dimension of the feature matrix to increase the efficiency of the model. Principal component analysis (PCA) is a statistical model which reduce large feature sets into smaller one while keeping most of the information intact. It transforms correlated features into some uncorrelated features called the principal component. This method is closed to a correlation technique that is applied to data with a wide difference of variance. PCA algorithm is mainly used for the algorithm with common share variance and focuses on a linear combination of a variable to extract maximum possible variance. In the PCA algorithm, eigenvector and eigenvalue are considered. The eigenvector shows a common variance and unique variance for producing correlation. Eigenvalue is the measure of all variance for a specific factor.

- Kaiser and Cumulative

The last layer of the proposed framework is to select the most significant permissions based on the eigenvalues and eigenvectors variables. Factor analysis is used in this step. Factor analysis is a statistical model to check variability between observed and unobserved factors. Kaiser's method sets a threshold between the maximum and minimum margin for the eigenvalue of 1. It means each eigenvalue above 1 demonstrates the desired factor and those below are not selected [27]. In this study, we applied Kaiser's method on the PCA results to find the most significant permissions.

Besides Kaiser, the cumulative technique was also used. Unlike Kaiser, there is no specific threshold for cumulative technique. Rather than a threshold, a percentage has been suggested. In most of the existing models, when 90% of the variance is reached, the model stopped. In our case, we also set the variance percentage to be 90% and applied the cumulative method on the PCA results to distinguish the most important permissions [28].

We compared Kaiser and cumulative results to obtain similar permission from the list. Similarly, this comparison has been done with Li [10] and Google dangerous permission list. The Final selected permissions are obtained from the intersection of all the results.

- Significant Permission Matrix

In the final step of the data processing, the desired feature matrix with the most significant permissions is formed. Also, the data label is added to the model to be used for classification in the machine learning and detection part.

V. Experimental Setting and Results

This section summarily discussed how the data are extracted and processed, the implementation that is done using machine learning approaches and the discussion on the evaluation of the proposed framework using some standard metrics.

A. Data Extraction

Permissions exist in the Android manifest file. To access permissions, extracting Android packages is required. For APK files in *ZIP* format, accessing them directly is not possible. There are two main methods to access the manifest file. First, extracting files manually and decoding AndroidManifest.xml file, and second, using a tool to extract them. In this work, the *Androguard* tool has been used. *Androguard* is a tool that extracts details from APK files [29]. When using this tool, we only need the permission extraction part. Thus, we extract all the required permissions and mapped them into the desired feature matrix to begin the data processing phase. The feature matrix store permissions as columns and applications as a row. The sample of the extracted data from the datasets is displayed in Table VIa.

B. Data Processing

Python is the main programming language used for the implementation. Scikit learn library is used for the PCA, Kaiser, and cumulative layers of our framework. Table VII shows the sample correlation among features. The main diagonal is 1 because the relation of each permission with itself is 1. Based on this matrix, each correlation variable is higher declared and these two permissions have much impact on the datasets.

C. Machine Learning

We used Scikit learn library to implement the malware detection model using SVM and decision tree algorithms. We used 80% of the datasets to train the model and 20% for the testing.

D. Experimental Results

We report the selected permission based on the multi-level permission extraction framework and its effectiveness on large datasets. To align with the main objective of this research, we are able to achieve a better optimized solution for large datasets as compared to the existing works.

- Significant Dangerous Permission

From our results, 16 permissions were highlighted as dangerous permissions. Table VI shows the list of high-risk permissions based on the proposed framework.

- Evaluation Metrics

The evaluation metrics considered in this research are precision, recall, and F-measure. Precision is the number of correct positive results by the number of positive predicted results. A recall is the number of correct positive results by the number of all datasets. F-measure is the balance between precision and recall rate. This rate tries to determine the accuracy of the classification. Mathematically, these metrics are defined in equations 5, 7, and 8.

$$Precision = \frac{True\ Positve\ Rate}{True\ Positive\ Rate+False\ Positive\ Rate} \tag{5}$$

$$Recall = \frac{True\ Positive\ Rate}{True\ Positve\ Rate+False\ Negative\ Rate} \tag{6}$$

$$F-Measure = \frac{2*Precision*Recall}{Precission+Recall} \tag{7}$$

Tables VIII and IX show the measurement metrics for our framework based on 20 selected permissions. From the tables, the input applications considered are 10000, 25000, and 60000. Although, the available datasets are higher than the number of applications, but we are able to get our desire threshold at 60000 based on the results.

Table VIII shows the performance metrics based on the SVM and Table IX contains the results based on the decision tree algorithm.

TABLE VI. DANGEROUS PERMISSION

CAMERA	RECEIVE_BOOT_COMPLETE
READ_CALL_LOG	ANSWER_PHONE_CALL
READ_CONTACTS	RECORD_AUDIO
READ_EXTERNAL_STORAGE	WRITE_SETTING
READ_PHONE_STAT	WRITE_EXTERNAL_STORAGE
READ_SMS	CHANGE_WIFI_STAT
WRITE_CONTACT	DISABLE_KEYGUARD
WRITE_SMS	SET_WALLPAPER

TABLE VIa : SAMPLE FEATURE MATRIX

	ACCESS	ACCESSORY	ACTIVITY	ANT	AUTHENTICATE	BATTERY	BIND	BLUETOOTH	BODY	BROADCAST	CALL	CAMERA	CHANGE	CLEAR	CONTENT	DEVICE	DISABLE	DOWNLOAD	EXPAND	FLASHLIGHT
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
5	1	1	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
9	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0
15	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0

	GET	GOOGLE	INSTALL	INTERACT	INTERNAL	INTERNET	KILL	MANAGE	MAPS	MODIFY	MOUNT	NFC	PACKAGE	PERSISTENT	PROCESS	READ	RECEIVE	RECORD	REGISTER	REGISTRATION
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
2	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
3	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
4	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
5	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
6	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
7	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0
8	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
9	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
11	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
12	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
13	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
14	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0
15	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0

	REORDER	RESTART	SEND	SERVICE	SET	STORAGE	SUBSCRIBED	SYSTEM	TASK	UA	UI	UNINSTALL	UPDATE	USE	VIBRATE	WAKE	WRITE
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	1	1
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
8	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	1	1
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
14	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

TABLE VII. CORRELATION BETWEEN PERMISSIONS

	ACCESSORY_FRAMEWORK	**ACCESS_COARSE_LOCATION**	**ACCESS_FINE_LOCATION**	**ACCESS_KEY_APP**	**ACCESS_LOCATION_EXTRA_COMMANDS**	**ACCESS_MOCK_LOCATION**	**ACCESS_NETWORK_STATE**	**ACCESS_WEB_PAGES**
ACCESSORY_FRAMEWORK	1	0.099614	0.097472	-0.00485	-0.01303	-0.00688	0.019474	-0.00485
ACCESS_COARSE_LOCATION	0.099614	1	0.717591	-0.04873	0.267477	0.141218	0.195498	-0.04873
ACCESS_FINE_LOCATION	0.097472	0.717591	1	-0.0498	0.205232	0.138181	0.199795	-0.0498
ACCESS_KEY_APP	-0.00485	-0.04873	-0.0498	1	-0.01303	-0.00688	0.019474	-0.00485
ACCESS_LOCATION_EXTRA_COMMANDS	-0.01303	0.267477	0.205232	-0.01303	1	-0.01848	0.052291	-0.01303
ACCESS_MOCK_LOCATION	-0.00688	0.141218	0.138181	-0.00688	-0.01848	1	0.027608	-0.00688
ACCESS_NETWORK_STATE	0.019474	0.195498	0.199795	0.019474	0.052291	0.027608	1	0.019474
ACCESS_WEB_PAGES	-0.00485	-0.04873	-0.0498	-0.00485	-0.01303	-0.00688	0.019474	1

TABLE VIII. PERFORMANCE METRICS BASED ON SVM

Number of applications	**10000**	**25000**	**60000**
Precision	98.20%	97.16%	95.17%
Recall	95.80%	93.75%	92.86 %
F-measure	96.98%	95.42%	94.00%

TABLE IX. PERFORMANCE METRICS BASED ON DECISION TREE

Number of applications	**10000**	**25000**	**60000**
Precision	98.99%	96.10%	92.11%
Recall	96.10%	93.20%	91.10 %
F-measure	97.53%	94.68%	91.60%

From the above tables, we say that the SVM performs better than the decision tree in all the measurement metrics. SVM shows promising results in false positive rate of 3.89 with 60000 applications. This is also shown in Fig. 5.

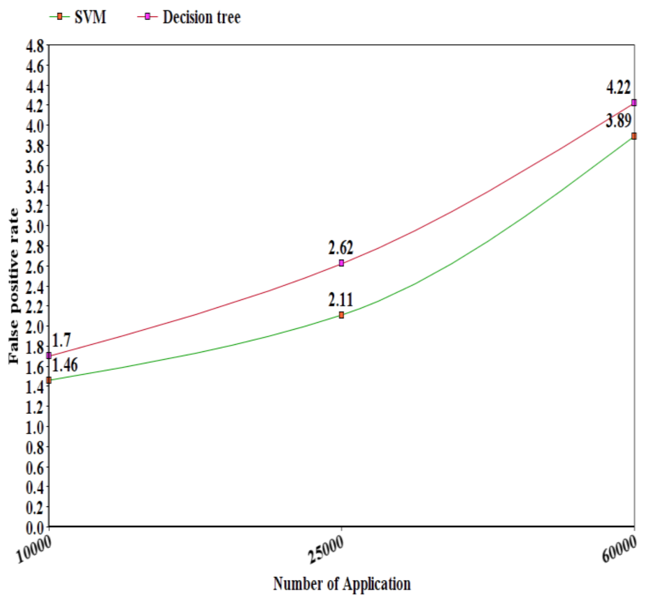

Fig. 5. False Positive Rate in SVM and Decision Tree.

- Discussion

Android malware detection is a crucial research area for the current digital world. Most of the handphones are running on Android. Due to the availability and wide usage of this platform, many attacks are carried out to take advantage of the user's personal data. Permission in Android is the main source of investigation on developer intent. Permission can be named as double edge blade which can be dangerous or helpful for users. In most cases users need to decide whether permission is granted or not. This decision is crucial when it comes to a wide range of applications on mobile phones. Google as the main developer of Android tried to facilitate this decision by adding extra labels to some permissions, however this barrier is still not enough.

The proposed framework successfully increased the detection rate by using only permissions. Although some researchers like [30] used an ensemble of features for detection but this method is complex and inefficient in a real-world implementation. In the work of [1], permission is used as the only feature for identification but their results show that as the data increases, the detection accuracy keep reducing. To maintain the detection accuracy when using a large number of datasets and one feature, we used different factor analysis methods such as Kaiser and cumulative to aggregate the significant permissions. Our results are compared with the results of [1] in Table X.

TABLE X. COMPARISON BETWEEN LI FRAMEWORK AND THE PROPOSED FRAMEWORK

Performance metrics	**Li Framework**		**Proposed Framework**	
	No. of Applications	Results	No. of Applications	Results
Precision	2650	98.83%	10000	98.20%
	5494	97.54%	25000	97.16%
	54694	95.5%	60000	95.17%
Recall	2650	94.4%	10000	95.80%
	5494	93.62%	25000	93.75%
	54694	92.17%	60000	92.86%
F-measure	2650	94.97%	10000	96.98%
	5494	95.54%	25000	95.42%
	54694	93.63%	60000	94.00%
False positive rate	2650	1.17%	10000	1.46%
	5494	2.36%	25000	2.11%
	54694	4.85%	60000	3.89%

During the analysis, some problems were faced. Some applications do not follow normal patterns. This anomaly can be code obfuscation or APK encryption. Some attackers encode the Android manifest file which makes it inaccessible for analysis. In this case, analysing the pattern of application in terms of network connection or requested permission while the application is running can help to distinguish malicious applications. But in most cases, this analysis is costly in terms of resource and time.

VI. Conclusion

Due to high demand and availability, the Android has become a famous platform for malicious activity. There are existing algorithms developed to avoid malware attacks on this platform but these algorithms are inefficient. Static, dynamic, and hybrid analysis are the three main techniques used to investigate malicious applications. The authors proposed a multi-level permission extraction framework to identify significant permissions to differentiate between normal and malicious applications in Android devices. The method used is based on static analysis as the researchers' focus is on the Android packaging file (APK) in a static environment. Permission was used as a feature to develop the proposed model and the model is able to achieve a better detection accuracy as compared to the existing works. To prune out unnecessary permissions, the researchers employed different mathematical steps in the proposed framework. The SVM and decision tree algorithms were used for the classification with different number of datasets. The results obtained are promising as with 60,000 applications, the model achieved 94% accuracy. This result is better when compared to other existing models. Hybrid method (a combination of static and dynamic permission analysis) would be adopted to see if better results can be achieved. Adding more features from applications and increasing the datasets will also be considered in our future work.

Acknowledgment

The authors would like to acknowledge the Universiti Putra Malaysia for supporting this research.

References

[1] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-An, and H. Ye, "Significant Permission Identification for Machine-Learning-Based Android Malware Detection," IEEE Trans. Ind. Informatics, Vol. 14, No. 7, Pp. 3216–3225, 2018, Doi: 10.1109/Tii.2017.2789219.

[2] J. Walls and K.-K. R. Choo, "A Review of Free Cloud-Based Anti-Malware Apps for Android," in 2015 IEEE Trustcom/BigDataSE/ISPA, 2015, pp. 1053–1058, doi: 10.1109/Trustcom.2015.482.

[3] Roman Unuchek, "Mobile malware evolution 2017," 2018.

[4] L. Dua and D. Bansal, "TAXONOMY: MOBILE MALWARE THREATS AND DETECTION TECHNIQUES," pp. 213–221, 2014, doi: 10.5121/csit.2014.4522.

[5] R. Zachariah, M. S. Yousef, and A. M. Chacko, "Android Malware Detection A Survey," no. Iccs, 2017.

[6] F. Wei, Y. Li, S. Roy, X. Ou, and W. Zhou, "Deep Ground Truth Analysis of Current Android Malware," Springer, Cham, 2017, pp. 252–276.

[7] E. Alpaydin, Introduction to machine learning. MIT Press, 2010.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436–444, May 2015, doi: 10.1038/nature14539.

[9] S. Makridakis, E. Spiliotis, and V. Assimakopoulos, "Statistical and Machine Learning forecasting methods: Concerns and ways forward," 2018, doi: 10.1371/journal.pone.0194889.

[10] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye, "Significant Permission Identification for Machine Learning Based Android Malware Detection," IEEE Trans. Ind. Informatics, vol. 3203, no. c, 2018, doi: 10.1109/TII.2017.2789219.

[11] N. Milosevic, A. Dehghantanha, and K. K. R. Choo, "Machine learning aided Android malware classification," Comput. Electr. Eng., vol. 61, pp. 266–274, 2017, doi: 10.1016/j.compeleceng.2017.02.013.

[12] N. McLaughlin et al., "Deep Android Malware Detection," Proc. Seventh ACM Conf. Data Appl. Secur. Priv. - CODASPY '17, pp. 301–308, 2017, doi: 10.1145/3029806.3029823.

[13] J. Mcgiff, W. G. Hatcher, J. Nguyen, W. Yu, E. Blasch, and C. Lu, "Towards Multimodal Learning for Android Malware Detection," 2019 Int. Conf. Comput. Netw. Commun. ICNC 2019, pp. 432–436, 2019, doi: 10.1109/ICCNC.2019.8685502.

[14] S. Badhani and S. K. Muttoo, "CENDroid—A cluster–ensemble classifier for detecting malicious Android applications," Comput. Secur., Apr. 2019, doi: 10.1016/J.COSE.2019.04.004.

[15] K.-K. R. C. &Ramlan M. Mohsen Damshenas, Ali Dehghantanha, "No Title," J. Inf. Priv. Secur., vol. 11, no. 3, pp. 141–157, 2015.

[16] E. M. B. Karbab, M. Debbabi, and D. Mouheb, "Fingerprinting android packaging: Generating DNAs for malware detection," DFRWS 2016 USA - Proc. 16th Annu. USA Digit. Forensics Res. Conf., vol. 18, pp. S33–S45, 2016, doi: 10.1016/j.diin.2016.04.013.

[17] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic android malware detection system with ensemble learning," IEEE Access, vol. 6, pp. 30996–31011, 2018, doi: 10.1109/ACCESS.2018.2844349.

[18] M. Park, J. Han, H. Oh, and K. Lee, "Threat Assessment for Android Environment with Connectivity to IoT Devices from the Perspective of Situational Awareness," Wirel. Commun. Mob. Comput., vol. 2019, pp. 1–14, Apr. 2019, doi: 10.1155/2019/5121054.

[19] J. Zhang, Z. Qin, H. Yin, L. Ou, and K. Zhang, "A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding," Comput. Secur., vol. 84, pp. 376–392, 2019, doi: 10.1016/j.cose.2019.04.005.

[20] R. Agrawal and R. S&ant, "Fast Algorithms for Mining Association Rules," 20th VLDB Conf. Santiago, Chile, 1994.

[21] M. Damshenas, A. Dehghantanha, K.-K. R. Choo, and R. Mahmud, "M0Droid: An Android Behavioral-Based Malware Detection Model," J. Inf. Priv. Secur., vol. 11, no. 3, pp. 141–157, 2015, doi: 10.1080/15536548.2015.1073510.

[22] A. Mahindru and P. Singh, "Dynamic Permissions based Android Malware Detection using Machine Learning Techniques," in Proceedings of the 10th Innovations in Software Engineering Conference on - ISEC '17, 2017, pp. 202–210, doi: 10.1145/3021460.3021485.

[23] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "AndroZoo," in Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16, 2016, pp. 468–471, doi: 10.1145/2901739.2903508.

[24] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto, "Stealth attacks: An extended insight into the obfuscation effects on Android malware," Comput. Secur., vol. 51, pp. 16–31, Jun. 2015, doi: 10.1016/j.cose.2015.02.007.

[25] Google, "The Structure of Android Package (APK) Files," Nov. 2010.

[26] R. Banchs, Text Mining with MATLAB. New York, NY: Springer New York, 2013.

[27] J. Ruscio and B. Roche, “Determining the number of factors to retain in an exploratory factor analysis using comparison data of known factorial structure.,” Psychol. Assess., vol. 24, no. 2, pp. 282–292, Jun. 2012, doi: 10.1037/a0025697.

[28] B. Williams, T. Brown Andrys Onsman, A. Onsman, T. Brown, P. Andrys Onsman, and P. Ted Brown, “Exploratory factor analysis: A five-step guide for novices Recommended Citation) &quot;Exploratory factor analysis: A five-step guide for novices Exploratory factor analysis: A five-step guide for novices,” This J. Artic. is posted Res. Online, vol. 8, pp. 2010–990399, 2012.

[29] S. H. Anthony Desnos, “Androguard,” 2011.

[30] W. Wang, Z. Gao, M. Zhao, Y. Li, J. Liu, and X. Zhang, “DroidEnsemble: Detecting Android Malicious Applications with Ensemble of String and Structural Static Features,” IEEE Access, pp. 1–1, 2018, doi: 10.1109/ACCESS.2018.2835654.