A Hybrid Approach for Android Malware Detection and Family Classification

Meghna Dhalaria, Ekta Gandotra*

Department of Computer Science and Engineering, Jaypee University of Information Technology, Waknaghat, Solan, HP (India)

Received 13 February 2020 | Accepted 29 May 2020 | Published 1 September 2020

Abstract

With the increase in the popularity of mobile devices, malicious applications targeting Android platform have greatly increased. Malware is coded so prudently that it has become very complicated to identify. The increase in the large amount of malware every day has made the manual approaches inadequate for detecting the malware. Nowadays, a new malware is characterized by sophisticated and complex obfuscation techniques. Thus, the static malware analysis alone is not enough for detecting it. However, dynamic malware analysis is appropriate to tackle evasion techniques but incapable to investigate all the execution paths and also it is very time consuming. So, for better detection and classification of Android malware, we propose a hybrid approach which integrates the features obtained after performing static and dynamic malware analysis. This approach tackles the problem of analyzing, detecting and classifying the Android malware in a more efficient manner. In this paper, we have used a robust set of features from static and dynamic malware analysis for creating two datasets i.e. binary and multiclass (family) classification datasets. These are made publically available on GitHub and Kaggle with the aim to help researchers and anti-malware tool creators for enhancing or developing new techniques and tools for detecting and classifying Android malware. Various machine learning algorithms are employed to detect and classify malware using the features extracted after performing static and dynamic malware analysis. The experimental outcomes indicate that hybrid approach enhances the accuracy of detection and classification of Android malware as compared to the case when static and dynamic features are considered alone.

Keywords

Android Malware, Dynamic Malware Analysis, Machine Learning, Static Malware Analysis.

DOI: 10.9781/ijimai.2020.09.001

I. Introduction

SMARTPHONES have become an open source platform for running different types of applications (apps) such as banking, lifestyles, gaming, education, etc. According to the site-worldwide mobile application, download of apps reached 205.4 billion in year 2018 and will increase continuously [1]. The fast growth in the smartphone industry has made lot of users to use smartphones to consume multiple services and access the Internet. The Android apps bring lot of comfort for our life by supporting persistent communication everywhere and also providing diverse functionalities. The expansion of Android apps plays a vital role for the progress of upcoming economy and mobile Internet.

The smartphones usually store user's private data such as messages, pictures and personal information etc. As a result, these smartphones become the target of attackers [2], [3]. Nowadays in smartphone industry, Android operating system (OS) has gained the highest position throughout the world. In 2018, the wide use of Android apps has resulted in an increase of Android malware (approximately 2.84 million) [4]. According to the report of McAfee, 31 million Android malware were found in 2018 and also shows that approximate 1.9 million new samples are identified every year [5]. As a result, it has become complicated to manually process large amount of Android malware samples. Thus, it becomes a most challenging task for antivirus companies to detect and classify malware. To evade the problem of handling large amount of malware samples manually and the malware obfuscation, the researchers start finding efficient techniques of Android malware detection and family classification.

* Corresponding author.

E-mail address: ekta.gandotra@gmail.com

The researchers are making use of several methods for detection of Android malicious apps. The traditional method to identify Android malware is relying on a signature based technique in which the signature of an app is matched with the already existing signatures present in the database. The major limitation of this technique is that it cannot identify unfamiliar malware. The ongoing research for detection and classification of malware is based on two methods i.e. static and dynamic malware analysis [2]. Static malware analysis method examines the code of the app to detect the malicious patterns without running the code [6]. It provides fast detection and high efficiency. But this method fails to identify the Android apps which make use of code obfuscation techniques [7]. The dynamic malware analysis method investigates the behavior of app while executing in a virtual environment. It is more efficient but this method is resource and time intensive. Moreover, this type of analysis is incapable to investigate all the execution paths. In order to strengthen the accuracy, the features acquired from both static and dynamic analysis can be integrated [8]. Moreover, there exists only limited benchmark datasets available publically to evaluate the proposed machine learning techniques.

In this paper, we have worked on both detection and family classification of Android malware. Here detection relates to a binary classification problem which consists of two classes "malware" and "benign" and family classification relates to the multiclass classification problem which consists of 13 malicious families. Android malware family signifies a group of malicious programs that share common behavior and are generated from the same source code. We propose a hybrid approach for detection and classification of Android malicious apps. It depends on the fusion of static and dynamic malware analysis. Initially, we perform static malware analysis for extracting static features based on API calls, command strings, permissions and intents. Then, we performed dynamic malware analysis to extract features using CuckooDroid [9]. CuckooDroid is an extension of cuckoo sandbox which is used for automatic analysis of Android suspicious files [10]. The features considered for dynamic malware analysis are based on cryptographic operations, dynamic permissions, information leaks and system calls. In order to strengthen the accuracy, we integrate the features acquired from both static and dynamic malware analysis. Considering the presence of irrelevant, noisy and redundant features, an information gain ranking algorithm is applied to extract the relevant features.

A. Research Contributions

The major contributions of the paper are as follows:

1. Two datasets i.e. binary and multiclass (family) classification datasets are created (using static and dynamic malware analysis) and shared publically on GitHub and Kaggle.
2. Feature selection method is used to choose the appropriate set of features for both the datasets.
3. The relevant features selected for both static and dynamic malware analysis are integrated.
4. Machine learning (ML) algorithms belonging to different categories are employed and evaluated on both the datasets for static, dynamic and integrated features.

B. Organization

The rest of the paper is structured as follows: section II summarizes the related work on classification and identification of Android malware. Section III describes the proposed methodology. Section IV demonstrates the experimental outcomes based on different evaluation parameters. Section V concludes the paper and provides future scope.

II. Related Work

In the literature, researchers have developed various novel techniques for identification and classification of Android malware using ML methods. Current malware identification methods fall under two categories i.e. static and dynamic malware analysis [11]. This section discusses the work associated with malware detection and classification based on static and dynamic malware analysis using ML methods.

A. Static Malware Analysis

The static malware analysis is the way to discover the malicious patterns in app by examining its code. In order to find out the malicious patterns [12], it uses disassemble techniques to decompile the app source code [13]. This subsection includes the research papers related to static malware analysis which focuses on detection and classification of Android malware.

Li et al. [14] suggested a malware identification system known as significant Permission Identification (SigPID). They build 3 levels of pruning by extracting permission data to determine the relevant permissions that can be to distinguish between malware and benign apps. The authors employed ML methods to classify the Android apps. The experimental results show that SigPID performs better with 93.62% of accuracy as compared to existing approaches. In [15], the authors suggested a highly efficient method to extract API calls, permission-rate, surveillance system events and permissions as features. They constructed a model based on ensemble Rotation Forest to identify whether an app is malicious or benign. The results demonstrate that the proposed approach obtained highest precision of 88.16% with 88.26% accuracy at the sensitivity of 88.40%. Yerima and Sezer [16] introduced a novel fusion technique (DroidFusion) which includes amalgamation of various ML techniques for improving accuracy. The DroidFusion creates a model by training classifiers and then they employed a feature ranking algorithm on the predictive accuracies in order to acquire a final classifier. The results indicate that DroidFusion is more superior than stacking ensemble method. In [17], the authors presented a multimodal deep learning based framework for the identification of Android malware. They extracted diverse features and refined these using similarity based or existence-based method. The results show that the accuracy obtained by the multimodal deep learning framework is 98%. Feizollah et al. [18] presented an analysis of the usefulness of intents for classifying the malicious apps. They reported that intents are more important feature than permissions for classification of malware. The results demonstrate that detection rate of intent and permission is 91% and 83% respectively. The authors also indicate that the detection accuracy of combined features is 95.5% which is higher than the individual features. In [19], the authors explored the risk based on permissions in Android apps. They applied T-test, correlation coefficient and mutual information to rank the specific permission according to their risk. Principal component analysis and sequential forward selection are employed to determine the subsets of risky permission. They evaluated the effectiveness of risky permission for detection of malapp with Decision Tree (DT) Support Vector Machine (SVM) and Random Forest (RF). The results indicate that the detection accuracy of malapp detector is 94.62% with 0.6 False Positive Rate (FPR). Dhalaria et al. [20] performed a comparative analysis between different base classifiers such as SVM, Logistic Regression (LR), Naive Bayes (NB) K-Nearest Neighbor (K-NN), DT, RF and ensemble techniques (Bagging, Stacking and Boosting). The experimental results demonstrate that the stacking ensemble technique found to be more superior then the base classifiers. Dhalaria et al. [21] employed a convolutional neural network (CNN) to classify Android malicious apps. The grayscale images of classes.dex and AndroidManifest.xml are created which are extracted from the Android package. The experimental results indicate that the classes.dex file performs better in comparison to AndroidManifest.xml.

The static malware analysis is quicker in analyzing the code but it fails against code obfuscation techniques and morphed malware. The dynamic malware analysis overwhelms the constraints of static malware analysis.

B. Dynamic Malware Analysis

It executes the samples in runtime environment such as an emulator and a virtual machine to track the behavior of the app. This section includes the literature on detection and classification of Android malware using dynamic malware analysis.

Cai et al. [22] presented a novel classification approach (DroidCat) which is based on dynamic analysis. The authors used a set of dynamic features such as method calls, app resources and Inter-Component Communication. The experimental outcomes indicate that DroidCat obtained 97% accuracy and F-measure for classifying the Android malicious apps. In [23], the authors proposed a dynamic analysis framework i.e. EnDroid which used different types of dynamic features for the identification of malware. They employed a chi-square algorithm to select the relevant features and applied an ensemble learning technique to differentiate between malware and benign apps. Das et al. [24] proposed the model named as frequency centric for feature construction using system calls to effectively identify the malware. The authors build a ML method using Multilayer Perceptron (MLP) in FPGA in order to train a classifier. They found that the proposed approach obtained low power consumption, fast detection and high accuracy. In [25], the authors addressed TaintDroid, a dynamic taint tracking which is proficient of continuously tracking various source of sensitive data. As a result, it provides security service firms seeking and essential input for Android users to identify malicious apps. Chen et al. [26] presented a framework which uses a classification scheme named as Model-Based Semi-Supervised (MBSS). The authors also compared their proposed approach with the existing approach such as K-NN, Linear Discriminant Analysis (LDA) and SVM. The results indicate that the proposed approach achieves 98% accuracy at very low FPR. In [27], the authors designed and implemented a dynamic analysis method named as DroidTrace. It examined the system calls which are executed in dynamic payloads. DroidTrace also carried out physical alteration to trigger numerous dynamic loading behaviors within an app.

The dynamic malware analysis can detect the unfamiliar malware that a static analysis cannot but it takes more time and resources. Moreover, it explores only a single execution path.

C. Hybrid Malware Analysis

Gandotra et al. [8] suggested that single approach either dynamic or static is not sufficient for accurately classifying the malware due to the obfuscation and execution stalling. To overcome this problem, the researchers have started to make use of a hybrid analysis approach. This section includes the work done in the field of hybrid malware analysis which focuses on detection and classification of Android malware.

Yuan et al. [28] introduced an engine named as DroidDetector which automatically characterized the app as either malware or benign. The authors extracted the features using static and dynamic analysis. The experimental results demonstrate that DroidDetector obtained highest accuracy 96.76% when compared with conventional ML techniques. In [29], the authors proposed the hybrid approach for identification of malware using static and dynamic analysis. They created the normal and malicious pattern sets by matching the pattern of benign and malware apps with each other. To determine the unknown app, the authors also compared these with both normal and malicious pattern sets offline. The results demonstrate that the proposed approach obtained better detection rate. Martin et al. [30] presented an OmniDroid dataset consisting of 22,000 malware and benign samples. They developed a framework for static and dynamic analysis of apps and applied ensemble learning classifiers for identification of malicious apps. In [31], the authors presented an Android Application Sandbox (AASandbox) which is capable to carry out both dynamic and static analysis to identify malicious apps. For providing distributed and fast detection, they deployed the detection algorithm and sandbox in the cloud. The results show that AASandbox is more efficient than antivirus apps available for Android OS.

From the literature survey, it is found that the hybrid approach is capable to classify the Android apps more accurately. Though, a lot of work has been reported in the literature on detection (binary classification) of Android apps using hybrid approach but the least focus has been paid on family classification of Android malware. Moreover, there exist only two benchmark datasets i.e. Malgenome [3] and Derbin [32] which have been made public over past few years. These datasets include old Android apps and were created in the years 2012 and 2014 respectively. But nowadays, evolving malwares are so sophisticated and complex that they cannot be recognized easily. This paper presents the approach used for creating our own datasets. These consist of recent Android apps and we have made these publically available on GitHub and Kaggle. These would help the research community to evaluate their proposed ML techniques for malware classification. Different machine learning algorithms are employed on these two datasets to perform binary and family classification of Android apps when both static and dynamic features are integrated.

III. Proposed Methodology

This section discusses the proposed methodology for detection and family classification of Android apps. It consists of three phases i.e. data collection, data preparation and detection & family classification. In the first phase, data is collected from various sources such as virusshare [33], apkmirror [34] and apkpure [35]. In the second phase, MD5 hash is applied to remove the duplicate apps and then these apps are examined using Avira Antivirus (AV) tool [36]. The static and dynamic malware analysis is performed to extract features from the Android apps. Static features are extracted using self-developed python script which uses multiple automated tools such as Baksmali Diassembler [37], String [38] and AXMLPrinter2 [39]. The features extracted using static malware analysis includes API calls, command string, permissions and intents. Dynamic features are extracted using CuckooDroid [9] which analyzes the behavior of app during runtime. The features extracted using dynamic malware analysis include dynamic permissions, cryptographic operations, information leaks and system calls. After feature extraction, an information gain feature ranking algorithm is employed in order to remove the noisy, irrelevant and redundant features. Various ML classifiers such as SVM, DT, RF, NB, K-NN PART and MLP are employed to identify and classify the Android apps. Fig. 1 shows the workflow of the proposed methodology.

A. Data Collection (Phase-I)

The initial phase of the proposed methodology is data collection. The Android apps are collected from multiple sources such as apkpure, apkmirror and virusshare. These apps are stored in Android application packages (.apk) file format. A total of 4400 recent Android apps are downloaded from these sources. The malicious apps are downloaded from virusshare after getting registered with their website and also getting permission from the administrator. The benign apps are collected from apkpure and apkmirror.

B. Data Preparation (Phase-II)

This subsection discusses various steps used for data preparation. These include removing duplicate applications, labelling, feature extraction and feature selection.

1. Removing Duplicate Applications

MD5 hash algorithm is employed on the collected Android apps to eliminate the duplicate ones. After removing the duplicates, we are left with 3547 Android apps.

2. Labelling

The unique Android apps obtained from the previous step are scanned using Avira Antivirus (AV) tool for labelling. After labelling, out of 3547 apps, 1747 are malicious and 1800 are benign. Furthermore, 1747 malicious apps are further labelled as 13 malware families as shown in Fig. 2.

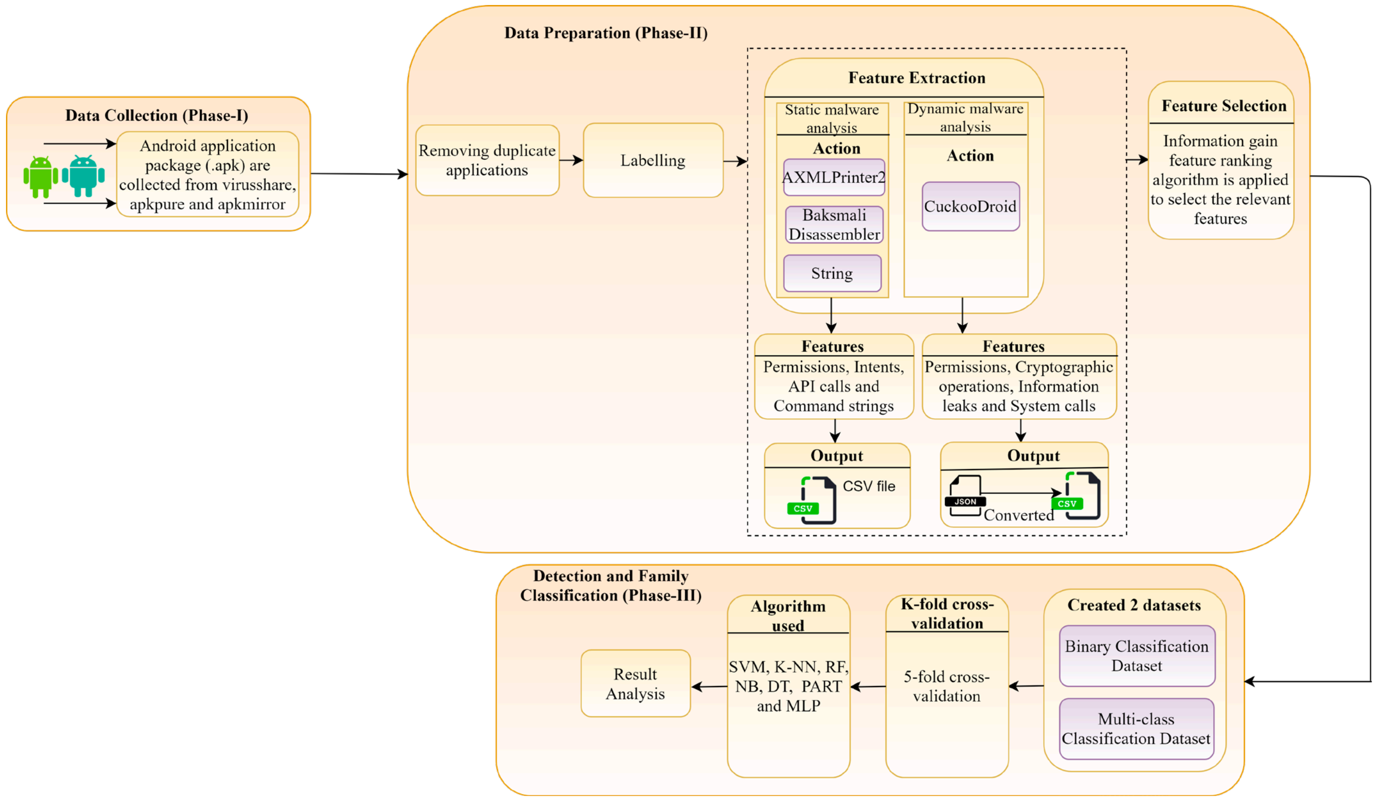

Data Collection (Phase-I)
Android application package (.apk) are collected from virusshare, apkpure and apkmirror
Data Preparation (Phase-II)
Removing duplicate applications
Labelling
Feature Extraction
Static malware analysis
Action
AXMLPrinter2
Baksmali Disassembler
String
Dynamic malware analysis
Action
CuckooDroid
Features
Permissions, Intents, API calls and Command strings
Features
Permissions, Cryptographic operations, Information leaks and System calls
Output
CSV file
Output
JSON
CSV
Converted
Feature Selection
Information gain feature ranking algorithm is applied to select the relevant features
Detection and Family Classification (Phase-III)
Created 2 datasets
Binary Classification Dataset
Multi-class Classification Dataset
K-fold cross-validation
5-fold cross-validation
Algorithm used
SVM, K-NN, RF, NB, DT, PART and MLP
Result Analysis

Fig. 1. Workflow of the proposed methodology.

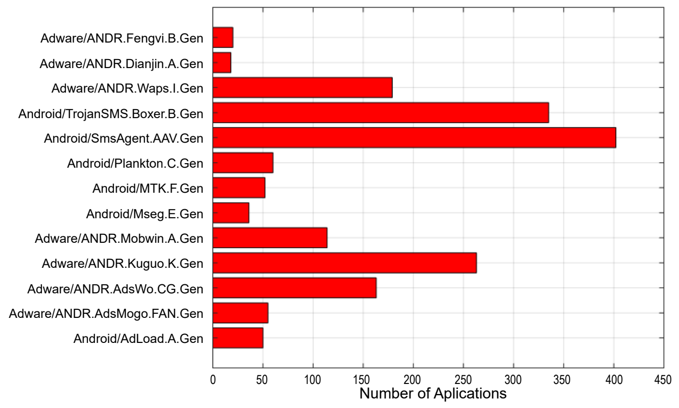


Fig. 2. Graphical representation of Android malware families.

3. Feature Extraction

Various features are extracted using static and dynamic malware analysis. In static malware analysis, we have extracted four different types of static features i.e. API calls, intents, permissions and command strings using self-developed python script which uses several automated tools such as Baksmali Disassembler, AXMLPrinter2 and string. In dynamic malware analysis, we have extracted four different types of dynamic features i.e. cryptographic operations, dynamic permissions, information leaks and system calls using CuckooDroid (Android malware analysis tool). The detailed description related to feature extraction using static and dynamic malware analysis is explained below.

a) Using Static Malware Analysis

It is performed without executing the code. It uses various disassemble techniques to decompile the app source code. To extract the static features, we developed a python script which uses various automated tools i.e. Baksmali Disassembler, AXMLPrinter2 and string. The features extracted for analysis using these tools are API calls, permissions, intents and command strings. The process of extracting features is shown in Fig. 3. The .apk file is saved in compressed zip format. To view the content of .apk file, we first need to unzip or unpack it. The .apk file consists of classes. dex file, Android Manifest file, res, lib and assets folder. Through this, we extracted four different types of static features using different static tools. Classes. dex file contains information about API calls, Android Manifest file contains information about permission and intents and the rest contains information about command strings. These features are selected on the basis of existing literature and the official site of Android which says that these specific features are more prominent in malicious applications [16], [40].

- **API calls**: It is used to interact with the device. These contain the method, classes and packages to help developers to build apps. The Android is based on java programming language and Java compiler converts the source code into java bytecode. It uses Dalvik Virtual Machine (DVM) after disassembling java bytecode, it gives information about packages, methods and classes. A total of 47 API calls are extracted using a self-developed python script after decompiling classes.dex with Baksmali Disassembler.
- **Permissions**: The main purpose of permissions is to secure the privacy of the users. The apps must request permission to access user sensitive information and system features. The system sometimes gives permission itself or could provoke users to accept the request. Permission is mainly declared in the AndroidManifest. xml. A total of 277 permissions are extracted using a self-developed python script after decompiling AndroidManifest.xml with AXMLPrinter2.
- **Command strings**: It is one of the static features which is used for identification of Android malware. It analyzes the command string which is present in lib, res, assets folder. A total of 6 command strings are extracted using a self-developed python script after

decompiling lib, res and assets with string.

- **Intents**: Intents are found in Manifest.xml. It infers the intentions of apps e.g. pick a contact, dial a number etc. Intents are extracted from manifest.xml after decompiling with AXMLPrinter2. A total of 22 intents are extracted using a self-developed python script after decompiling AndroidManifest.xml with AXMLPrinter2. Table I lists some of the examples of static features considered under these four categories.

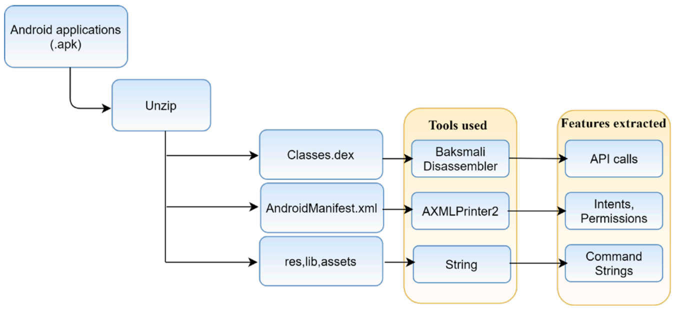


Fig. 3. Process of extracting static features.

b) Using Dynamic Malware Analysis

It is performed while executing the code in the runtime environment. The runtime behavior information of the apps is obtained using the open source dynamic analysis tool named as CuckooDroid. It is an extension of cuckoo sandbox, the open source software for executing and analyzing the apps. It automatically executes and analyzes files and collects the information of the file at runtime. CuckooDroid is liable for handling the Android emulator and produce report at the termination of analysis. Cuckoo's infrastructure consists of a guest machine (i.e. the virtual machine that carry out analysis) and the host machine (i.e. the management software). The host runs the main components of the sandbox that controls the whole analysis process, whereas the guest machine is the isolated environment where the Android malware samples are carried out. The guest machine consists of Linux virtual machines that run Android emulator, which is monitored by the machinery module. The main work of Android emulator is to carry out the execution of apps, collect information and report it back to CuckooDroid. Every Android malicious file is run until all processes are finished or a timeout of 180 seconds is reached which means an Android sample is given a maximum of 180 seconds for analysis. After the analysis of particular sample is over, the results are compiled in JSON format. We need a guest machine which is to be rooted Android Virtual Device (AVD) with xposed framework [41] and with its two module i.e. Emulator Anti-Detection and Droidmon. Python 2.7 is used to run the analyzer code and python agent on guest machine. The role of the python agent is for analysing code, receiving APK file, and carrying out the analysis. The python analyzer executes apps, send screenshots back to host, send dropped files back to host. It is liable for terminating the analysis and sending back some log file to host. After the complete procedure, the log reports are collected which is in the Java Script Object Notation (JSON) format. The reports produced by Cuckoo Droid for different apps are then parsed and saved to the database in CSV format using Python script. Afterwards, these are used for detection and classification of malware. The process of extracting dynamic features is shown in Fig. 4. The features extracted for analysis are cryptographic operations, information leaks, dynamic permissions and system calls.

These features are selected on the basis of existing literature and the official site of Android which says that these specific features are more prominent in malicious applications [22], [40], [42]. The detailed description of these four features is explained as follows:

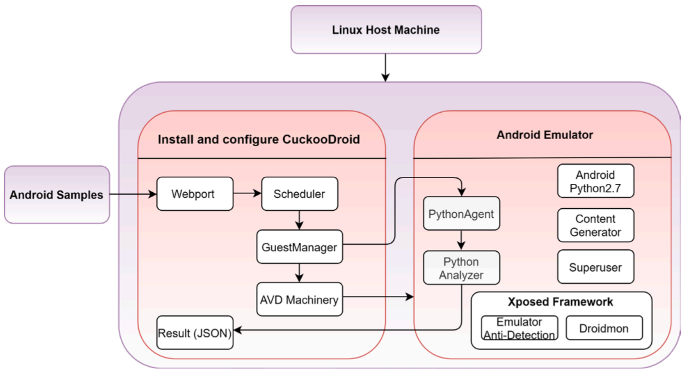


Fig. 4. Process of extracting dynamic features.

- **Cryptographic operations**: Malware accepts these operations to target premium sms number, encrypt root exploits, malicious payload etc. To distinguish various cryptographic behaviors, these features are formed as <action>_<algorithm >. Here <action> includes various operations like key generation, decryption and encryption and the <algorithm> includes various cryptographic algorithms. A total of 79 cryptographic operations are extracted using CuckooDroid.
- **Dynamic permissions**: It is considered as one of the important dynamic features to analyze the behavior of apps. Dynamic permissions are those permissions which are executed at the runtime environment. A total of 71 dynamic permissions are extracted at runtime using CuckooDroid.
- **Information leaks**: Confidential and personal data has newly gained more attention. Malware usually vigorously harvests numerous data on contagious devices, such as contact information, IMEI, SMS contents, credential information related to social network and banking etc. The collected data may be used to make profits, keep track on users and acquire authorized account etc. These features are defined as <source>_<sink>. Here <source> includes operations gaining confidential data and the <sink> includes operations leaking confidential data. A total of 123 information leaks are extracted at runtime using CuckooDroid.
- **System Calls**: It is one of the most important dynamic features of Android app. It is an efficient feature for intrusion detection in a mobile device. Through system calls, Android apps take services of the kernel. The kernel offers useful functions to apps such as device security, process related to operations and power management etc. These malware usually invokes sigprocmask, getuid, ptrance to affect the execution of other apps. A total of 50 system calls are extracted at runtime using CuckooDroid. Table II lists some of the examples of dynamic features considered under these four categories.

After performing static and dynamic malware analysis, a total of 352 static and 323 dynamic features are extracted from all the Android apps considered in this work. Thus, we have come up with two datasets. First is a binary classification dataset consisting of 1747 malicious and 1800 benign apps. Second is a multiclass classification dataset consisting of 1747 malicious apps belonging to 13 malware families. Both these datasets are made public on GitHub and Kaggle (Link: https://github.com/Meghna-Dhalaria/Android-malware-dataset) and (Link: https://www.kaggle.com/meghnadhalaria/android-malware-detection-and-classification) respectively.

4. Feature Selection

It is also known as attribute selection. It is used for dimensionality reduction which helps in choosing relevant features. Irrelevant and redundant features can decrease the quality of the classification model

TABLE I. Examples of Static Features Considered

Features	Number of features	Examples	Feature value
API Calls	47	onserviceConnected, Ljavax.crypto.spec.SecretKeySpec, getBinder, android.os.Binder, Ljava.net.URLDecoder, ServiceConnection, KeySpec, Ljava.lang.Class.getMethods	If an API call (out of 47) is existing in the classes.dex then the value of that feature is set to 1 otherwise 0.
Permissions	277	GET_TASKS, READ_PHONE_STATE, WRITE_EXTERNAL_STORAGE, RECEIVE_BOOT_COMPLETE, READ_SMS, SYSTEM_ALERT_WINDOW, RECEIVE_SMS, ACCESS_NETWORK_STATE	If a permission (out of 277) is existing in the Manifest.xml file then the value of that feature is set to 1 otherwise 0.
Command Strings	6	Chown, /system/bin, mount, /system/app, remount	If a command string (out of 6) is existing in the *res, lib, assets* folder then the value of that feature is set to 1 otherwise 0.
Intents	22	CALL_BUTTON, SET_WALLPAPER, NEW_OUTGOING_CALL, SCREEN_OFF, PACKAGE_CHANGED, ACTION_SHUTDOWN, BATTERY_LOW	If an intent (out of 22) is existing in the Manifest.xml file then the value of that feature is set to 1 otherwise 0.

TABLE II. Examples of Dynamic Features Considered

Features	Number of features	Examples	Feature value
Cryptographic Operations	79	Decryption_AES, encryption_AES, keyalgo_AES	If a cryptographic operation (out of 79) is present in JSON file then the value of that feature is set to 1 otherwise 0.
Dynamic Permissions	71	AUDIO_FILE_ACCESS, ACCESS_ GOOGLE_ PASSWORDS, WRITE_CONTACT_DATA, READ_CONTACT_DATA	If a dynamic permission (out of 71) is present in JSON file then the value of that feature is set to 1 otherwise 0.
Information Leaks	123	IMEI_File, IMSI_Network, IMSI_File, PHONE_NUMBER_File, IMEI_Network	If an information leak (out of 123) is present in JSON file then the value of that feature is set to 1 otherwise 0.
System Calls	50	ptrace, recvfrom, sigprocmask, write, wait4, sendto, getpid, read, recvmsg, chmod, sendmsg	If a system call (out of 50) is present in JSON file then the value of that feature is set to 1 otherwise 0.

and the accuracy. Higher dimensional datasets required more space and computation time [43]. Selecting the relevant features will help in reducing the space and time complexity and also help in increasing the accuracy. In this work, we have employed an information gain feature ranking algorithm [44] to select the relevant features for better detection and classification of Android malware. Information gain calculates the quantity of information provided about the class. It makes use of entropy to compute the homogeneity of samples. The entropy $H(X)$ of the dataset (having c number of classes) is calculated as given in equation (1).

$$H(X) = \sum_{i=1}^{c} -p_i \log_2 p_i \quad (1)$$

Where p_i is the probability of class i in the dataset X. The dataset is then split on the different attributes A. The entropy for a dataset with respect to attribute A i.e. $H(X, A)$ is calculated using equation (2).

$$H(X, A) = \sum_{k \in A} P(c) H(c) \quad (2)$$

Here k represents the possible values of the attribute A.

Information gain achieved by an attribute is expressed as shown in equation (3). Greater the Information Gain (IG) of a particular feature, more important the feature is.

$$IG = H(X) - H(X, A) \quad (3)$$

The information gain method assigns rank and weight to each feature. We have not considered the attributes with zero weight. Thus out of 352 features, we are left with 110 static features for binary classification dataset (named as Dataset-1) and 47 static features for family classification dataset (named as Dataset-2). Fig. 5 and Fig. 6 show the top 20 selected attributes for detection (Dataset-1) and family classification (Dataset-2) datasets respectively.

The datasets created using dynamic malware analysis consist of 323 features. Out of 323 features, we are left with 99 dynamic features in Dataset-1 and 35 features in Dataset-2. Fig. 7 and Fig. 8 show the top 20 selected dynamic features for detection (Dataset-1) and family classification (Dataset-2) datasets respectively.

The summary of both the datasets i.e. Dataset-1 and Dataset-2 before and after feature selection is given in table III. Fig. 9 shows the various steps for preparing these two datasets.

TABLE III. Description of Dataset (Where, # Stands for Number of)

Dataset Name	#Benign apps	#Malicious apps	#Feature extracted		#Feature selected	
			Static	Dynamic	Static	Dynamic
Dataset-1	1800	1747	352	323	110	99
Dataset-2	-----	1747 (with 13 families)	352	323	47	35

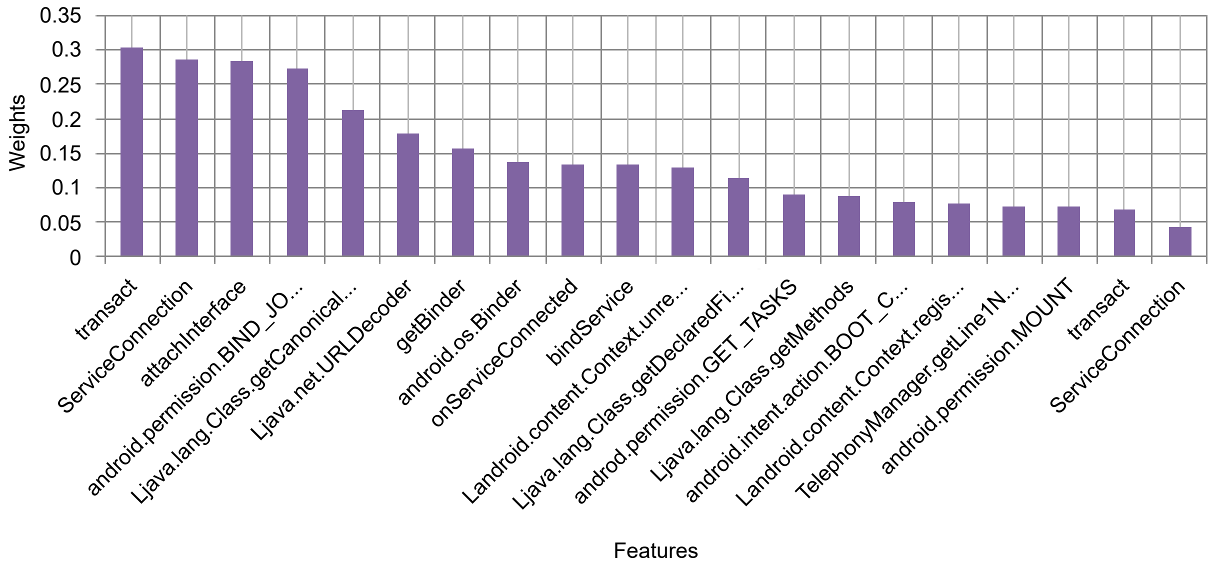


Fig. 5. Top 20 selected static features for detection dataset (Dataset-1).

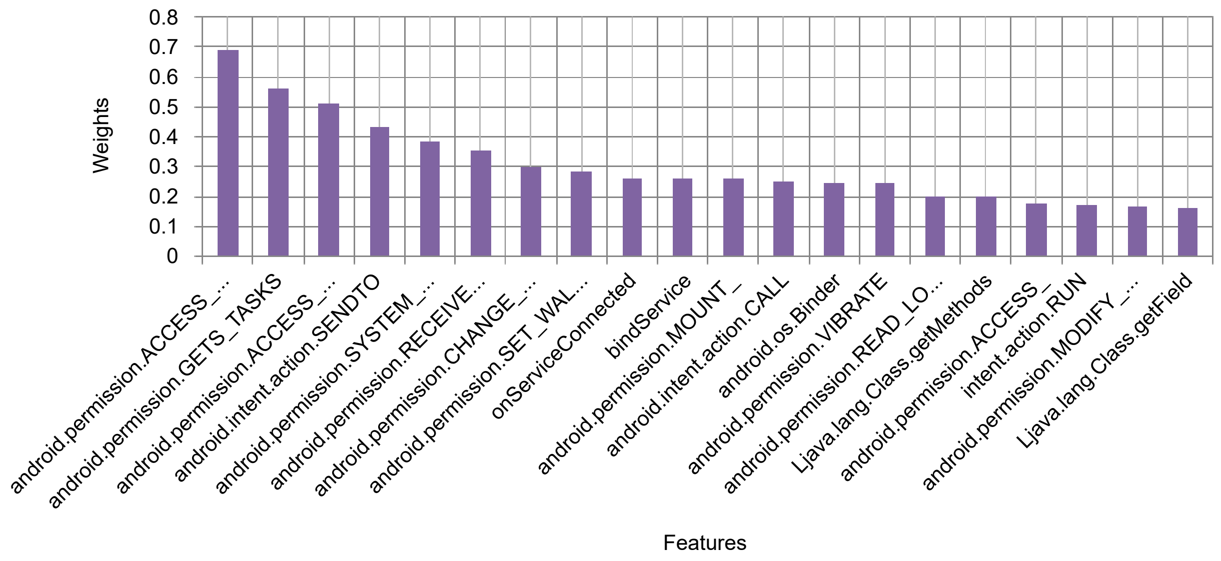


Fig. 6. Top 20 selected static features for family classification dataset (Dataset-2).

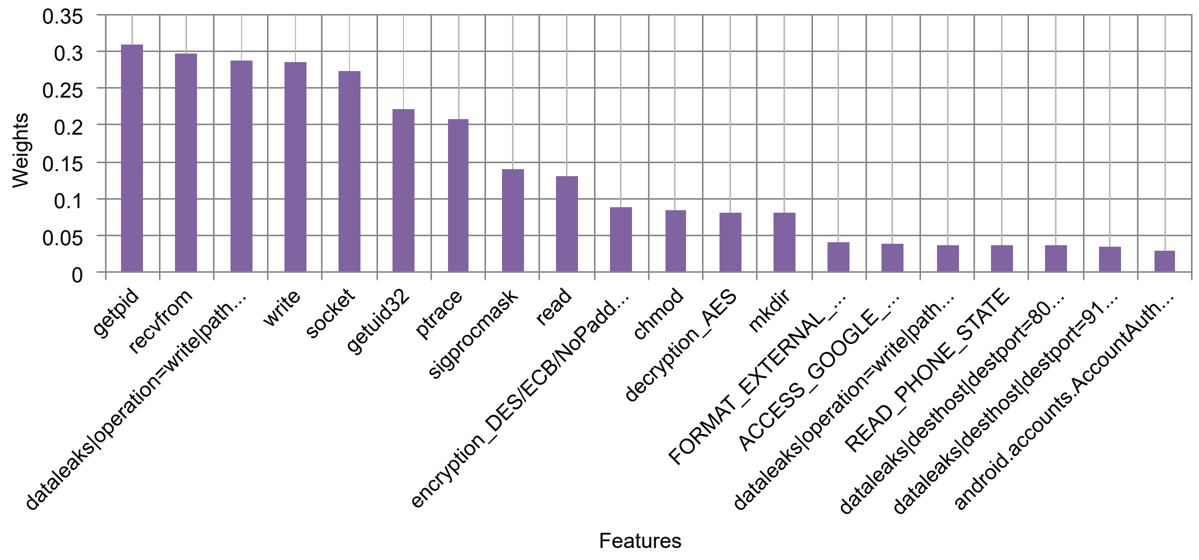


Fig. 7. Top 20 selected dynamic features for detection dataset (Dataset-1).

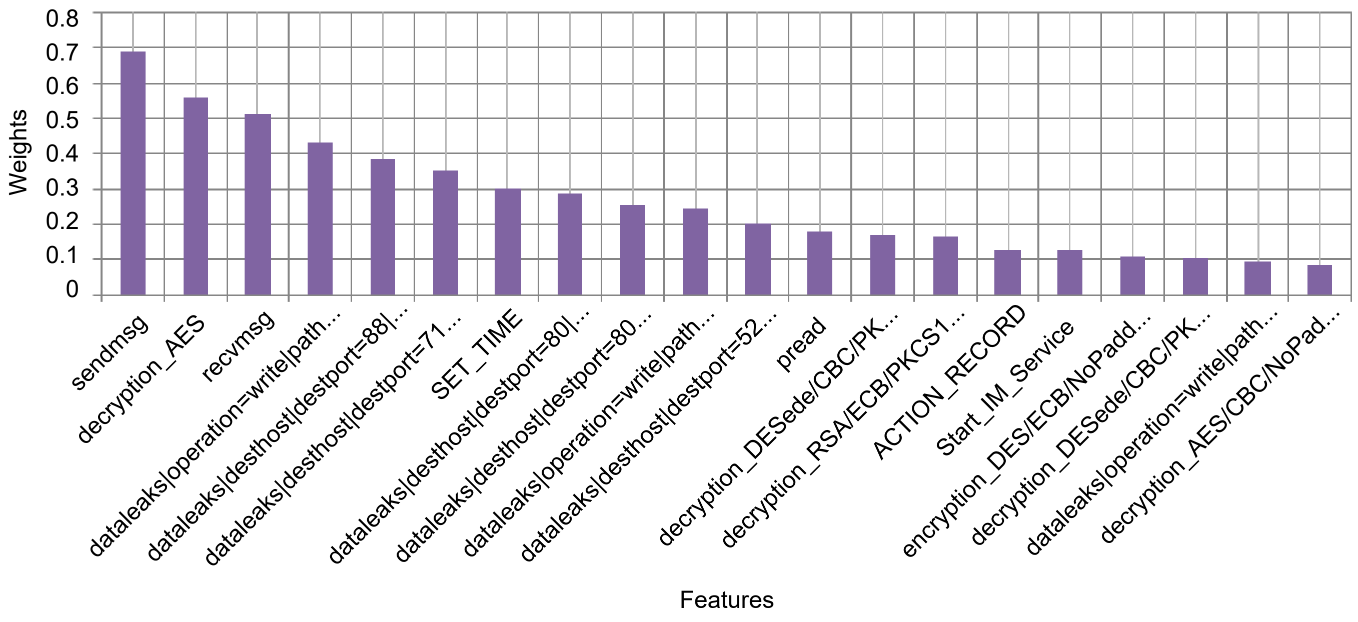


Fig. 8. Top 20 selected dynamic features for family classification dataset (Dataset-2).

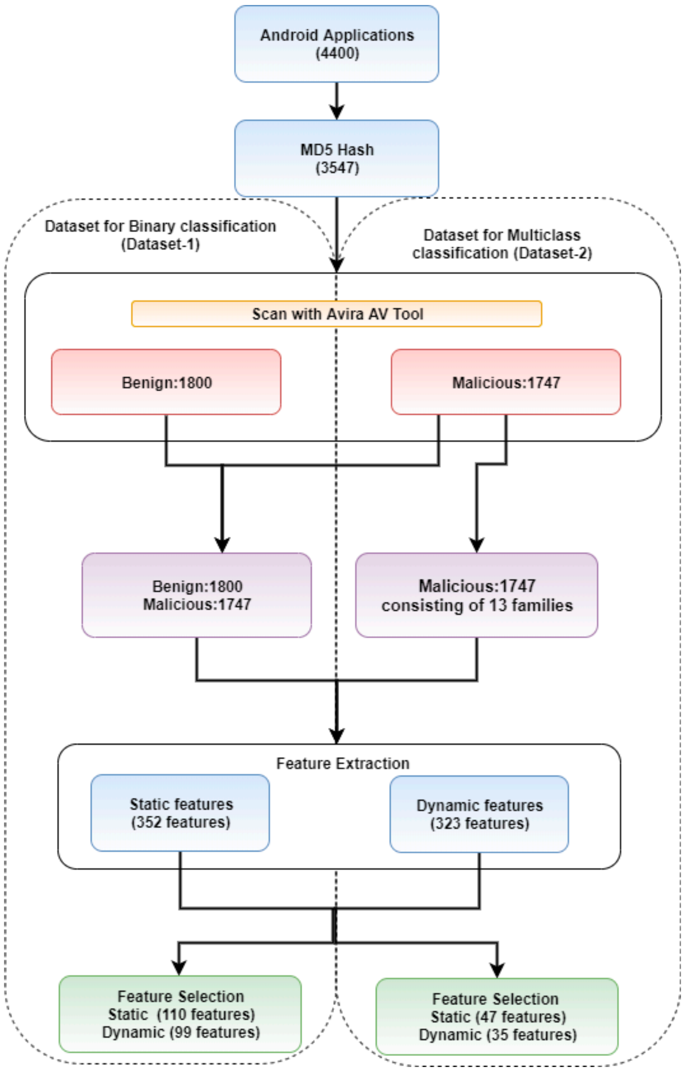

Android Applications (4400)
MD5 Hash (3547)
Dataset for Binary classification (Dataset-1)
Dataset for Multiclass classification (Dataset-2)
Scan with Avira AV Tool
Benign:1800
Malicious:1747
Benign:1800 Malicious:1747
Malicious:1747 consisting of 13 families
Feature Extraction
Static features (352 features)
Dynamic features (323 features)
Feature Selection Static (110 features) Dynamic (99 features)
Feature Selection Static (47 features) Dynamic (35 features)

Fig. 9. Process of Data Preparation.

C. Detection and Family Classification (Phase-III)

Various ML algorithms i.e. SVM, RF, DT, NB, K-NN, PART and MLP are used to build models for detection and classification of Android malware. These models are trained using 5-fold cross validation, in which the whole dataset is divided into 5 equal parts. Four parts are used to train the model and the remaining part is used for testing at every run. This section provides the brief introduction of ML algorithms and the evaluation parameters used for evaluating these algorithms.

1. Machine Learning Algorithms

The various ML algorithms used in this work are as follows:

- K-NN is one of the easiest supervised learning methods. It is also called as lazy learner [45]. This method does not depend upon the structure of data, whenever the new instance arises; it finds the closest training samples to the new instance by using distance measures such as Euclidean distance, Manhattan distance. At the end, by using the majority voting concepts it finds the class of the new instance.
- SVM is a method [46] which divides the data using a hyperplane. It acts like a decision boundary. It randomly draws the hyperplane and then computes the distance between the hyperplane and the closest data points (also called as support vector). It attempts to identify the optimal hyperplane that maximizes the margin.
- RF is an ensemble learning technique which involves a large number of individual decision trees that act as an ensemble [47]. Every decision tree produces a classification for input data and then RF collects the classification and illustrates the result based on majority voting.
- The structure of DT is like a tree, where non-leaf or internal node demonstrates a test on an attribute, topmost node represents the root node, terminal or leaf node holds a class label and the branch of the tree demonstrates the results of the test. In this work, we have used C4.5 algorithm to classify Android malware [48].
- The concept of NB is based on Bayes theorem. It forecasts the class membership probabilities i.e. the probability that a given tuples relates to an individual class. It is used for both binary and multiclass classification problems [49].

- PART is a partial decision tree algorithm. It is a separate and conquer rule learner. This technique produces sets of rules known as decision list. A new sample is compared to each rule and then the sample is assigned the class of the first matching rule [50].
- Multilayer Perceptron (MLP) is also called as Multilayer Neural Networks [51]. It consists of an input layer, an output layer and the hidden layer. It has various output units. The units of the hidden layer become input for the next layer. Semwal et al. [52], [53] worked in the field of different classification problems using deep learning techniques such as DNN based classifier and ANN. In [54], the authors [54] worked in the Extreme Machine Learning (ELM) for classification and prediction of gait data. In our work, we applied MLP for detection and classification of Android malware. We run the MLP for hidden layer h=3 and h=5 for Dataset-1 and Dataset-2 respectively. The activation function used for Dataset-1 and Dataset-2 are sigmoid and Softmax respectively. The learning rate is considered to be as 0.3. Fig. 10 shows the general framework of backpropagation based on neural network [53].

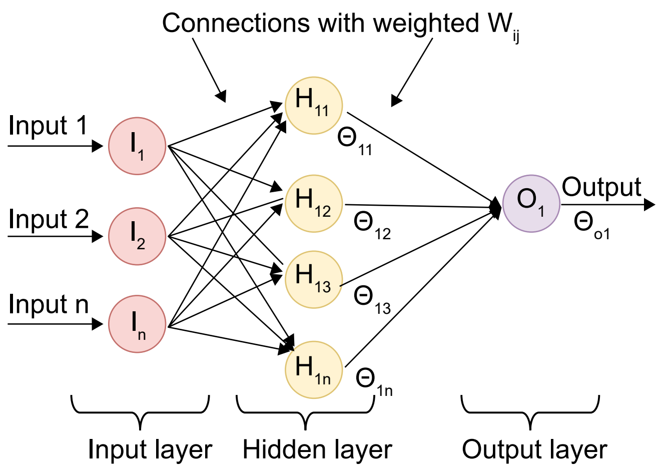


Fig. 10. General framework of backpropagation based on neural network [53].

The algorithm first initializes the weights to all nodes and then calculates the net input and output. It calculates the error rate and propagates it back. At the end, it updates the bias and weights and run the loop until the error becomes below the threshold.

2. Evaluation Parameters

The performances of the classifiers are assessed on the basis of various metrics such as precision, true positive rate (TPR), F-measure, false positive rate (FPR), Matthews correlation coefficient (MCC) and Area under curve (AUC) [55]. These performance metrics are defined using true negative (TN), false positive (FP), false negative (FN) and true positive (TP).

- **TPR**: It is also known as recall or sensitivity. It is defined as the ratio of true positive cases divided by the total number of actual positive cases. It is computed as shown in equation (4).

$$TPR = \frac{TP}{TP+FN} \tag{4}$$

- **FPR**: It is the ratio of false positive cases divided by total number of actual negative cases. It is computed as given in equation (5).

$$FPR = \frac{FP}{TN+FP} \tag{5}$$

- **Precision**: It is defined as the ratio of actual true predictive instances divided by the total number of true cases. It is computed as shown in equation (6).

$$Precision = \frac{TP}{TP+FP} \tag{6}$$

- **F-measure**: It signifies the harmonic mean of recall and precision. It is calculated as shown in equation (7).

$$F-measure = \frac{2\times(Precision\times Recall)}{(Precision+Recall)} \tag{7}$$

- **Accuracy**: It is the ratio of true positive and true negative instances divided by the total number of instances. It is calculated as shown in equation (8).

$$Accuracy = \frac{TP+TN}{TP+FP+FN+TN} \tag{8}$$

- **MCC**: It is used to measure the quality of binary classification algorithms. Its value lies between -1 to +1. Here -1 means inverse prediction and +1 means a perfect prediction. It is calculated as shown in equation (9).

$$MCC = \frac{TP\times TN-FP\times FN}{\sqrt{(TP+FN)(TP+FP)(TN+FP)(TN+FN)}} \tag{9}$$

- **AUC curve**: It is one of the most significant parameters to measure the performance of classification models. It represents the measure of the separability.

IV. Experimental Results

This section describes the experimental results based on static, dynamic and the hybrid features. Seven different ML technique are used which are run on python 3.7 under Intel Core i5 processor, 64 bit with 8GB RAM. We conducted the experiments using 5-fold cross validation method and evaluated the ML techniques on the basis of various evaluation parameters like TPR, F-measure, Accuracy, FPR, Precision, AUC and MCC.

A. Classification Results Based on Static Features

Seven ML algorithms are used to detect and classify malware on detection (Dataset-1) and family classification (Dataset-2). These algorithms are carried out in python script through *sklearn* [56] library.

Table IV demonstrates the evaluation results of ML techniques on static malware analysis for Dataset-1. It shows that RF gives the best accuracy of 96.50% followed by K-NN and MLP with accuracy as 95.74% and 95.71% respectively.

Fig. 11 shows the comparison of different classifiers based on accuracy and MCC of static features for Dataset-1. It indicates that RF performs better in comparison to other classifiers. The accuracy and MCC obtained by RF is 96.50% and 0.933 respectively.

Table V shows the evaluation results of ML techniques using static features for family classification on Dataset-2. It is found that RF algorithm gives better accuracy i.e. 86.72% followed by SVM and DT which gives and accuracy of 85.86% and 84.77% respectively. The TPR, precision and F-measure obtained by RF is 0.867, 0.870 and 0.866 respectively which are better results than those obtained by other classifiers.

Fig. 12 shows the comparative analysis of different classifiers based on accuracy for Dataset-2. The maximum accuracy of 86.72% is obtained by RF. This value is much smaller than the results obtained in static malware analysis for detection of malware in case of binary classification.

TABLE IV. Classification Results Using Static Features for Dataset-1

Classifiers	TPR	FPR	Precision	F-measure	MCC	AUC	Accuracy (%)
SVM	0.943	0.057	0.943	0.943	0.887	0.943	94.33
DT	0.950	0.050	0.950	0.950	0.901	0.970	95.03
NB	0.874	0.124	0.878	0.874	0.752	0.948	87.42
RF	0.965	0.035	0.965	0.965	0.933	0.990	**96.50**
K-NN	0.957	0.042	0.958	0.957	0.915	0.989	95.74
PART	0.950	0.050	0.950	0.950	0.900	0.975	94.98
MLP	0.957	0.043	0.957	0.957	0.914	0.986	95.71

TABLE V. Classification Results Using Static Features for Dataset-2

Classifier	TPR	FPR	Precision	F-measure	AUC	Accuracy (%)
SVM	0.859	0.023	0.863	0.857	0.962	85.86
DT	0.848	0.023	0.852	0.847	0.949	84.77
NB	0.751	0.032	0.792	0.756	0.967	75.10
RF	0.867	0.024	0.870	0.866	0.982	**86.72**
K-NN	0.845	0.024	0.847	0.843	0.966	84.48
PART	0.840	0.024	0.842	0.839	0.947	84.02
MLP	0.830	0.026	0.832	0.830	0.964	82.99

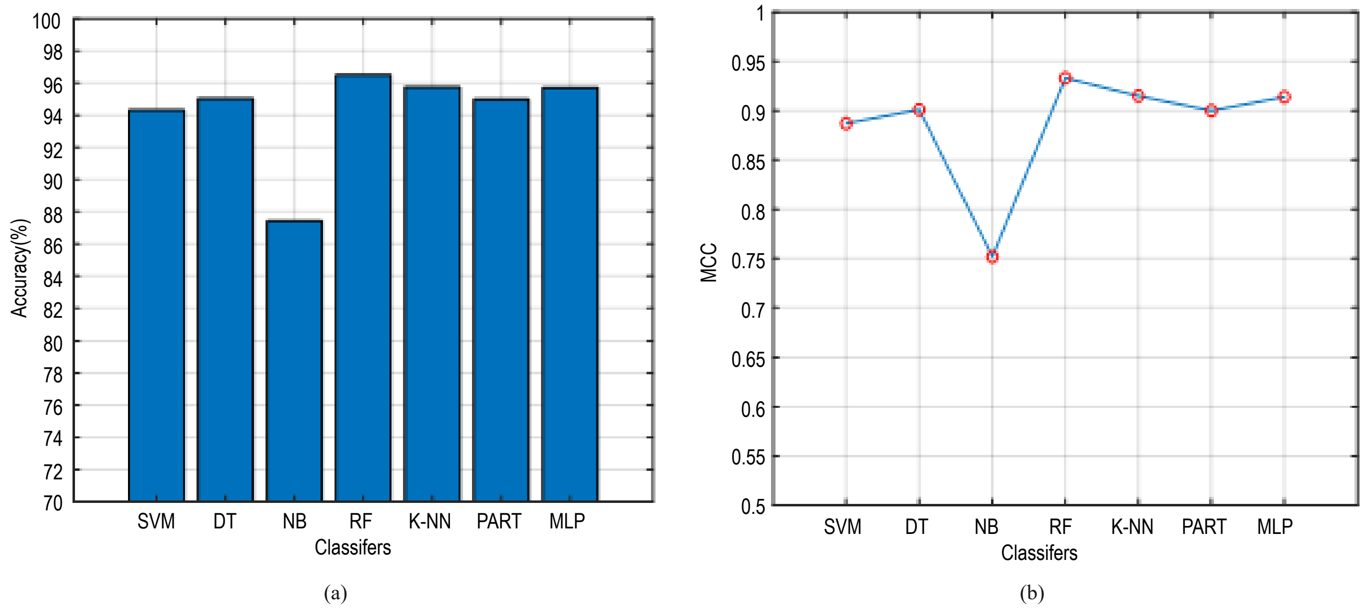


Fig. 11. Comparison of different classifiers based on (a) Accuracy (b) MCC using static features for Dataset-1.

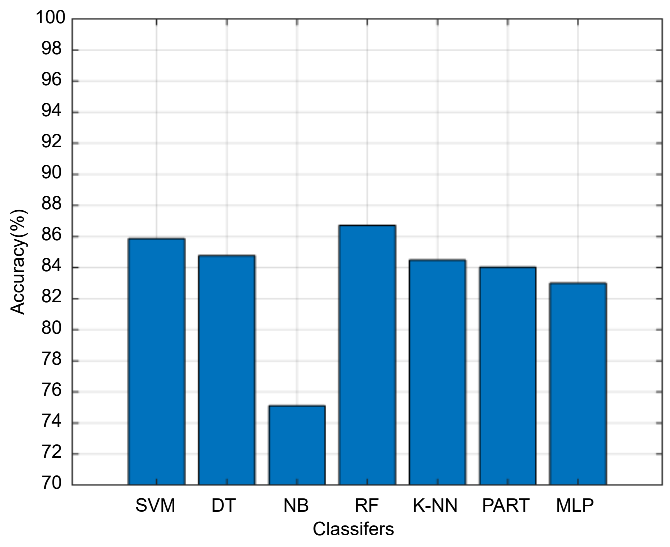


Fig. 12. Comparison of different classifiers based on accuracy using static features for Dataset-2.

B. Classification Results Based on Dynamic Features

The static malware analysis is quicker in analyzing the code but it fails against code obfuscation techniques and morphed malware. So to overcome this problem, we considered the dynamic features for better detection and classification of malware. Seven ML algorithms are used to detect and classify malware on detection (Dataset-1) and family classification (Dataset-2) datasets.

Table VI shows the evaluation results of ML techniques on dynamic malware analysis for malware detection (binary classification) on Dataset-1. Among all these classifiers, RF is found to be more superior and accurate than other classifiers. The accuracy acquired by RF is 97.01% followed by SVM and MLP with 96.53% and 96.53% respectively.

Fig. 13 shows the comparative analysis of different classifiers based on accuracy and MCC using dynamic features for Dataset-1. It indicates that RF performs better in comparison to other classifiers. The accuracy and MCC obtained by RF is 97.01% and 0.940 respectively.

Table VII shows the evaluation results of ML techniques on dynamic malware analysis for family classification on Dataset-2. Among all these classifiers, RF is found to be more superior and accurate than other classifiers. The accuracy obtained by RF is 88.60% followed by SVM and DT with 86.85% and 84.25% respectively. The TPR, precision and F-measure obtained by RF is 0.886, 0.888 and 0.885 respectively which are better values than those obtained by other classifiers.

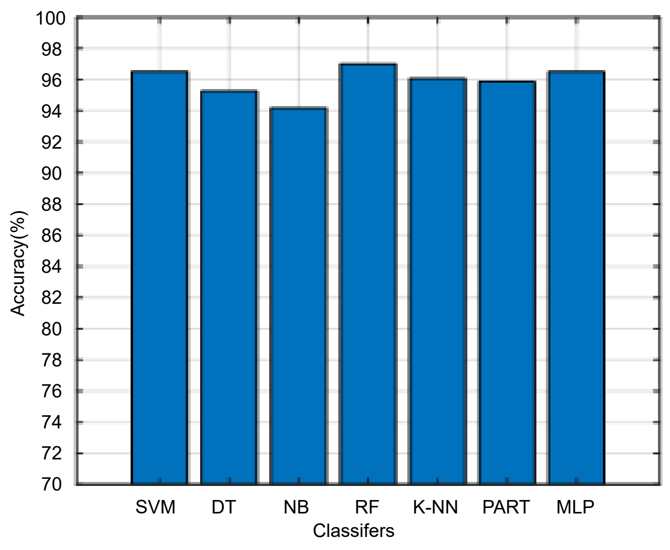
Fig. 14 shows the comparative analysis of different classifiers based on accuracy using dynamic features for Dataset-2. The maximum accuracy of 88.60% is obtained by RF. This value is much smaller than the results obtained in dynamic malware analysis for detection of malware (binary classification).

TABLE VI. Classification Results Using Dynamic Features for Dataset-1

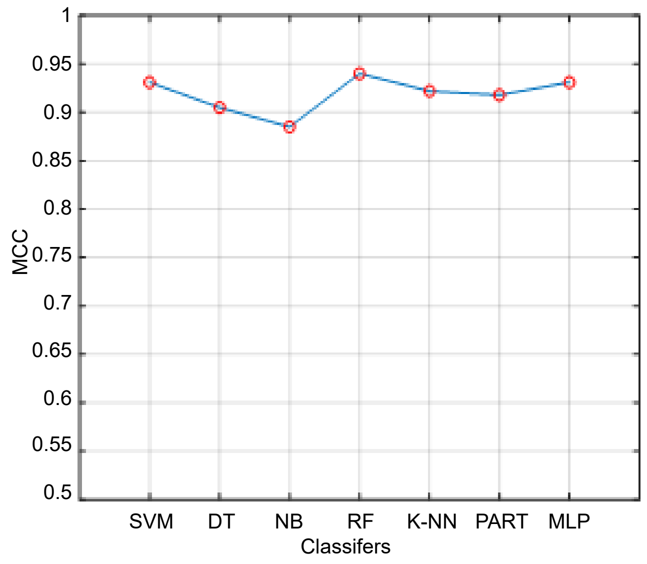
Classifier	TPR	FPR	Precision	F-measure	MCC	AUC	Accuracy (%)
SVM	0.965	0.035	0.965	0.965	0.931	0.965	96.53
DT	0.953	0.048	0.953	0.953	0.905	0.973	95.26
NB	0.942	0.057	0.943	0.942	0.885	0.989	94.19
RF	0.970	0.030	0.970	0.970	0.940	0.996	**97.01**
K-NN	0.961	0.039	0.961	0.961	0.922	0.990	96.08
PART	0.959	0.041	0.959	0.959	0.918	0.970	95.88
MLP	0.965	0.035	0.965	0.965	0.931	0.988	96.53

TABLE VII. Classification Results Using Dynamic Features for Dataset-2

Classifier	TPR	FPR	Precision	F-measure	AUC	Accuracy (%)
SVM	0.864	0.021	0.871	0.866	0.985	86.85
DT	0.843	0.026	0.843	0.841	0.947	84.25
NB	0.800	0.029	0.805	0.795	0.951	79.96
RF	0.886	0.018	0.888	0.885	0.991	**88.60**
K-NN	0.839	0.025	0.842	0.837	0.967	83.91
PART	0.841	0.026	0.838	0.836	0.950	84.08
MLP	0.829	0.027	0.828	0.825	0.947	82.88



(a)



(b)

Fig. 13. Comparison of different classifiers based on (a) Accuracy (b) MCC using dynamic features for Dataset-1.

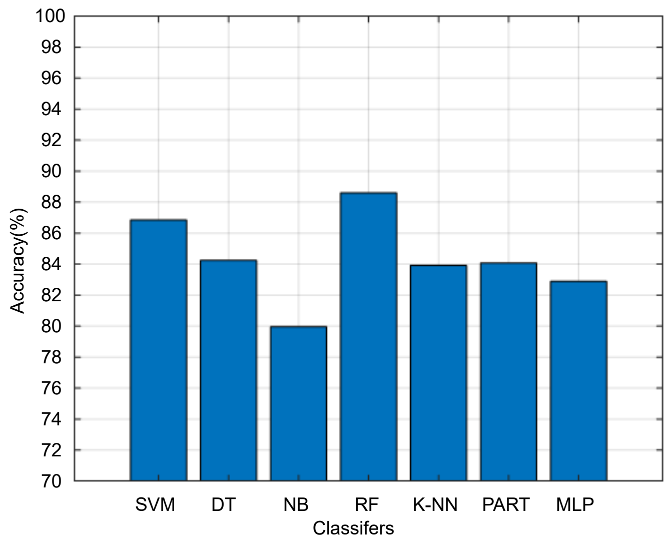


Fig. 14. Comparison of different classifiers based on accuracy using dynamic features for Dataset-2.

C. Classification Results Based on Integrated Features

Single approach either static or dynamic is inadequate for correctly classifying the malware due to the obfuscation and execution stalling. So to overcome this problem, we make use of a hybrid analysis approach. We integrated the features obtained from both static and dynamic malware analysis. Seven ML algorithms are used to detect and classify malware on detection (Dataset-1) and family classification (Dataset-2) datasets.

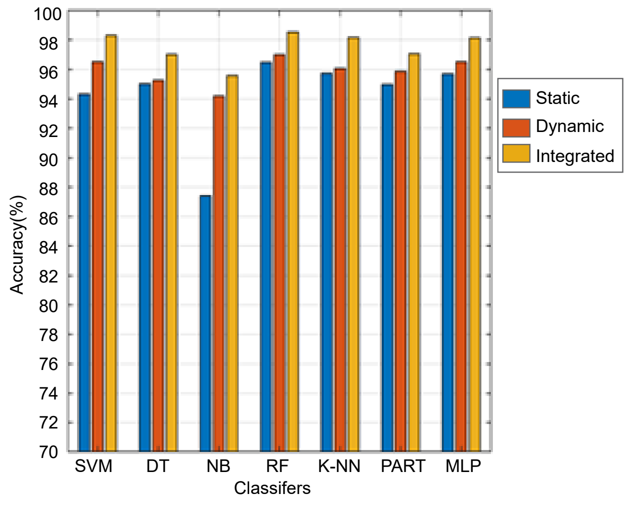
Table VIII shows the evaluation results of ML techniques on integrated features for Dataset-1. Among all these classifiers, RF is found to be more superior and accurate than other classifiers. The accuracy acquired by RF is 98.53% followed by SVM and K-NN with 98.30% and 98.16% respectively.

Table IX shows the evaluation results of ML techniques on integrated features for family classification for Dataset-2. Among all these classifiers, RF is found to be more superior and accurate than other classifiers. The accuracy acquired by RF is 90.10% followed by SVM and K-NN with 87.06% and 85.40% respectively. The TPR, precision and F-measure obtained by RF is 0.901, 0.902 and 0.901 respectively which are better results than those of other classifiers.

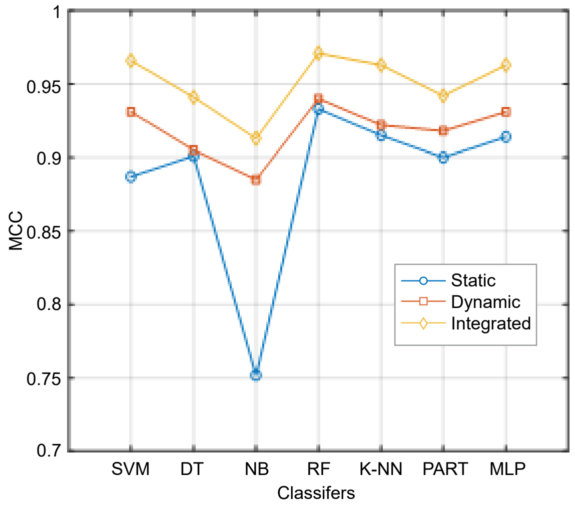
Fig. 15 shows the accuracy and MCC comparison of seven classifiers with respect to various approaches considered in our experiment for Dataset-1. It is clear from table VIII that there is an improvement in the accuracy and MCC for all the classifiers when the static and dynamic features are integrated. It means that using both static and dynamic features together helps for better detection and classification of the Android malware.

TABLE VIII. Classification Results Using Integrated Features for Dataset-1

Classifier	TPR	FPR	Precision	F-measure	MCC	AUC	Accuracy (%)
SVM	0.983	0.017	0.983	0.983	0.966	0.983	98.30
DT	0.970	0.030	0.970	0.970	0.941	0.980	97.03
NB	0.956	0.043	0.957	0.956	0.913	0.993	95.60
RF	0.985	0.015	0.985	0.985	0.971	0.999	**98.53**
K-NN	0.982	0.018	0.982	0.982	0.963	0.994	98.16
PART	0.971	0.029	0.971	0.971	0.942	0.983	97.09
MLP	0.981	0.019	0.981	0.981	0.963	0.993	98.13



(a)



(b)

Fig. 15. Comparison of different classifiers based on (a) Accuracy (b) MCC using static, dynamic and integrated features for Dataset-1.

TABLE IX. Classification Results Using Integrated Features for Dataset-2

Classifier	TPR	FPR	Precision	F-measure	AUC	Accuracy (%)
SVM	0.870	0.020	0.875	0.871	0.987	87.06
DT	0.846	0.024	0.851	0.845	0.949	84.60
NB	0.783	0.027	0.814	0.784	0.970	78.30
RF	0.901	0.016	0.902	0.901	0.995	**90.10**
K-NN	0.854	0.022	0.857	0.854	0.966	85.40
PART	0.833	0.024	0.837	0.833	0.946	83.34
MLP	0.845	0.024	0.847	0.845	0.963	84.48

TABLE X. Classification Results of Best Classifier Using Static, Dynamic And Integrated Features for Dataset-1 And Dataset-2

Dataset	Classifier	Approach	TPR	FPR	Precision	F-measure	MCC	Accuracy (%)
		Static	0.965	0.035	0.965	0.965	0.933	96.50
Dataset-1	**RF**	**Dynamic**	0.970	0.030	0.970	0.970	0.940	97.01
		Integrated	0.985	0.015	0.985	0.985	0.971	**98.53**
		Static	0.867	0.024	0.870	0.866	--	86.72
Dataset-2	**RF**	**Dynamic**	0.886	0.018	0.888	0.885	--	88.60
		Integrated	0.901	0.016	0.902	0.901	--	**90.10**

* MCC -- not applicable for multiclass dataset i.e. Dataset-2.

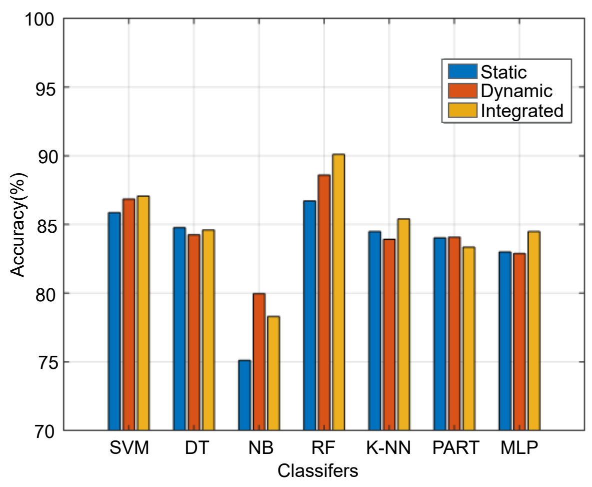


Fig. 16. Comparison of different classifiers based on accuracy using static, dynamic and integrated features for Dataset-2.

Fig. 16 demonstrates the comparison of seven classifiers on the basis of accuracy with respect to various approaches considered in our experiments for Dataset-2. It shows that for all the classifiers except NB and PART, the integrated approach performs better as compared to the cases when the static and dynamic features are considered alone. We are not able to achieve a good accuracy for the malware classification dataset (Dataset-2). It might be due to the imbalanced number of apps in different families.

Table X shows the comparison of static, dynamic and integrated approach for the best classifier i.e. RF for both the datasets i.e. Dataset-1 and Dataset-2. The results indicate that the integrated approach is found to be more appropriate for detection and classification of malware for both the datasets. The accuracy achieved by RF in case of Dataset-1 and Dataset-2 is 98.53% and 90.10% respectively. The overall performance shows that the integrated approach is more suitable in detection and classification of Android malware.

V. Conclusion and Future Work

This paper presented a hybrid approach which extracts different types of features using static and dynamic malware analysis to detect and classify Android malware. We created our own two datasets for detection (dataset-1) and family classification (dataset-2) of Android malware. Both datasets consist of 352 static features and 323 dynamic features. These datasets are made publically available on GitHub and Kaggle with the aim to help researchers and anti-malware tool creators for enhancing or developing new techniques and tools for detecting and classifying Android malware. The significance of the datasets makes it appropriate to be used as benchmark to test new techniques. We employed the information gain feature selection algorithm to eliminate noisy and irrelevant features. Through this algorithm, we selected 110 and 47 static features in Dataset-1 and Dataset-2 respectively and 99 and 35 dynamic features in Dataset-1 and Dataset-2 respectively. The features with zero weights are not considered here. Various ML classifiers are applied to detect and identify Android malware. The experimental results indicate that the hybrid approach obtains better detection and classification performance as compared to the cases when static and dynamic features are considered alone. For dataset-1, RF provides the accuracy of 96.5% when only static features are considered and 97.01% when only dynamic features are considered. For dataset-2, RF provides accuracy of 86.72% when only static features are considered and 88.6% when only dynamic features are considered. RF provides the highest accuracy in the hybrid approach (when both static and dynamic features are integrated) for both Dataset-1 and Dataset-2 i.e. 98.53% and 90.1% respectively.

In real world scenario, the malware classification problem is a data imbalance problem as there exist more examples of benign applications as compared to the malicious ones. In future, we will focus on this issue while using deep learning and big data tools [57] to classify the Android malware applications.

References

[1] StatistaReport. Accessed: December. 2019. [Online]. Available: http://www.statista.com/statistics/266488/forecast-of-mobile-appdownloads/.

[2] A. M. Memon, and A. Anwar, "Colluding apps: tomorrow's mobile malware threat," IEEE Security & Privacy, vol. 13 no. 6, pp. 77–81, 2015.

[3] Y. Zhou, and X. Jiang, "Dissecting Android malware: characterization and evolution," in IEEE Symposium in Security and Privacy, 2012, pp. 95–109.

[4] Future-Trends-of-Android-Malware-Growth. Accessed: December. 2019. [Online]. Available: https://www.researchgate.net/figure/Future-Trends-of-Android- Malware-Growth.

[5] McAfee Labs. (2018) Threat Predictions Report, McAfee Labs, Santa Clara, CA, USA.

[6] D. Barrera, H. G. Kayacik, P. C. V. Oorschot, and A. Somayaji, "A methodology for empirical analysis of permission-based security models and its application to Android," in Proc. of 17th ACM Conf. Computer and Communications Security, CCS 10, 2010, pp. 73–84.

[7] S. Singla, E. Gandotra, D. Bansal, and S. Sofat, "Detecting and classifying morphed malwares: A survey," International Journal of Computer Applications, vol. 122, no. 10, 2015.

[8] E. Gandotra, D. Bansal, and S. Sofat, "Malware analysis and classification: A survey," Journal of Information Security, vol. 5, no. 02, p. 56, 2014.

[9] CuckooDroid. Accessed: October. 2019. [Online]. Available: https://cuckoo-droid.readthedocs.io/en/latest/installation/.

[10] E. Gandotra, D. Bansal, and S. Sofat, "Malware intelligence: beyond malware analysis," International Journal of Advanced Intelligence Paradigms, vol. 13, no. 1-2, pp. 80-100, 2019.

[11] G. Suarez-Tangil, J. Tapiador, P. Peris-Lopez, and A. Ribagorda, "Evolution, detection and analysis of malware for smart devices," IEEE Communications Surveys & Tutorials, vol. 16, no. 2, pp. 961–987, 2013.

[12] S. Moghaddam, and M. Abbaspour, "Sensitivity analysis of static features for Android malware detection," in Electrical Engineering (ICEE), Tehran, Iran, 2014, pp. 920–924.

[13] Q. Li, and X. Li, "Android malware detection based on static analysis of characteristic tree," in Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Xian, China, 2015, pp. 84-91.

[14] J. Li, L. Sun, Q. Yan, Z. Li, W. Srisaan, and Y. Heng, "Significant permission identification for machine-learning-based android malware detection," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3216-3225, 2018.

[15] H. J. Zhu, Z. H. You, Z. X. Zhu, W. L. Shi, X. Chen, and L. Cheng, "DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model," Neurocomputing, vol. 272, pp. 638-646, 2018.

[16] S. Y. Yerima, and S. Sezer, "Droidfusion: A novel multilevel classifier fusion approach for android malware detection," IEEE transactions on cybernetic, vol. 49, no. 2, pp. 453-466, 2018.

[17] T. Kim, B. Kang, M. Rho, S. Sezer, and E. G. Im, "A multimodal deep learning method for Android malware detection using various features," IEEE Transactions on Information Forensics and Security, vol. 14, no. 3, pp. 773-788, 2018.

[18] A. Feizollah, N. B. Anuar, R. Salleh, G. S. Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," Computers & Security, vol. 65, pp. 121-134, 2017.

[19] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," IEEE Transactions on Information Forensics and Security, vol. 9, no. 11, pp. 1869-1882, 2014.

[20] M. Dhalaria, E. Gandotra, and S. Saha, "Comparative Analysis of Ensemble Methods for Classification of Android Malicious Applications," in advances in Computing and Data Sciences, M. Singh, P. K. Gupta, V. Tyagi, J. Flusser, T. Oren and R. Kashyap, Eds. Singapore: Springer International Publishing, 2019, pp. 370-380.

[21] M. Dhalaria and E. Gandotra, "Convolutional Neural Network for Classification of Android Applications Represented as Grayscale Images," International Journal of Innovative Technology and Exploring Engineering (IJITEE), vol. 8, no. 12S, pp. 835-843, 2019.

[22] H. Cai, N. Meng, B. Ryder, and D. Yao, "Droidcat: Effective android malware detection and categorization via app-level profiling," IEEE Transactions on Information Forensics and Security, vol. 14, no. 6, pp. 1455-1470, 2018.

[23] 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.

[24] S. Das, Y. Liu, W. Zhang, and M. Chandramohan, "Semantics-based online malware detection: Towards efficient real-time protection against malware," IEEE transactions on information forensics and security, vol. 11, no. 2, pp. 289-302, 2015.

[25] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "TaintDroid: an information-flow tracking system for real-time privacy monitoring on smartphones," ACM Transactions on Computer Systems (TOCS), vol. 32, no. 2, p. 5, 2014.

[26] L. Chen, M. Zhang, C. Y. Yang, and R. Sahita, "Semi-supervised classification for dynamic Android malware detection," arXiv preprint arXiv: 1704.05948, 2017.

[27] M. Zheng, M. Sun, and J. C. S. Lui, "DroidTrace: A ptrace based Android dynamic analysis system with forward execution capability," in international wireless communications and mobile computing conference (IWCMC), Nicosia, Cyprus, 2014, pp. 128-133.

[28] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," Tsinghua Science and Technology, vol. 21, no. 1, pp. 114-123, 2016.

[29] F. Tong, and Z. Yan, "A hybrid approach of mobile malware detection in Android," Journal of Parallel and Distributed computing, vol. 103, pp. 22-31, 2017.

[30] A. Martín, R. L. Cabrera, and D. Camacho, "Android malware detection through hybrid features fusion and ensemble classifiers: The AndroPyTool framework and the OmniDroid dataset," Information Fusion, vol. 52, pp. 128-142, 2019.

[31] T. Bläsing, L. Batyuk, A. D.Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," in 5th International Conference on Malicious and Unwanted Software,

Nancy, Lorraine, France, 2010, pp. 55-62.

[32] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. E. R. T. Siemens, “Drebin: Effective and explainable detection of android malware in your pocket,” In Ndss, vol. 14, pp. 23-26, 2014.

[33] Virusshare. Accessed: March. 2019. [Online]. Available: https://virusshare.com/.

[34] APKMirror. Accessed: March. 2019. [Online]. Available: https://www.apkmirror.com/.

[35] Apkpure. Accessed: March. 2019. [Online]. Available: https://apkpure.com/.

[36] Avira. Accessed: April. 2019. [Online]. Available: https://www.avira.com/.

[37] W. Enck, D. Octeau, P. D. McDaniel, and S. Chaudhuri, “A study of android application security,” In USENIX security symposium, vol. 2, p. 2, 2011.

[38] E. Gandotra, D. Bansal, and S. Sofat, “Tools & Techniques for Malware Analysis and Classification,” International Journal of Next-Generation Computing, vol. 7, no. 3, 2016.

[39] Android4me: J2ME port of Google’s Android (2011) https://code.google.com/p/android4me/downloads/list.

[40] Android Developers. Accessed: May. 2019. [Online]: Available: https://developer.android.com/guide/topics/manifest/permissionelement.

[41] Xposed module repository. Accessed: May. 2019. [Online]. Available: http://repo.xposed.info/module/de.robv.android.xposed.installer.

[42] S. Malik, and K. Khatter, “System call analysis of android malware families,” Indian Journal of Science and Technology, vol. 9, no. 21, 2016.

[43] B. Chizi, and O. Maimon, “Dimension reduction and feature selection,” in Data mining and knowledge discovery handbook, O. Maimon and L. Rokach, Eds. Boston MA: Springer, 2009, pp. 83-100.

[44] J. Han, J. Pei, and M. Kamber, “Data mining: concepts and techniques,” Elsevier, 2011.

[45] G. Shakhnarovish, T. Darrell, and P. Indyk, “Nearest-neighbor methods in learning and vision,” In MIT Press, 2005, p. 262.

[46] Keerthi, S. Sathiya, and E. G. Gilbert, “Convergence of a generalized SMO algorithm for SVM classifier design,” Machine Learning, vol. 46, no. 1-3, pp. 351-360, 2002.

[47] A. Liaw, and M. Wiener, “Classification and regression by randomForest,” R news, vol. 2, no. 3, 2002, pp. 18-22.

[48] J. R. Quinlan, “The Morgan Kaufmann Series in Machine Learning,” San Mateo, 1993.

[49] P. Domingos, and M. Pazzani, “On the optimality of the simple Bayesian classifier under zero-one loss,” Machine learning, vol. 29, no. 2-3, pp. 103-130, 1997.

[50] F. Eibe, and I. H. Witten, “Generating Accurate Rule Sets Without Global Optimization,” In: Fifteenth International Conference on Machine Learning, 1998, pp. 144-151.

[51] S. B. Joo, S. E. Oh, T. Sim, H. Kim, C. H. Choi, H. Koo, and J. H. Mun, “Prediction of gait speed from plantar pressure using artificial neural networks,” Expert Systems with Applications, vol. 41, no. 16, pp. 7398-7405, 2014.

[52] V. B. Semwal, K. Mondal, and G. C. Nandi, “Robust and accurate feature selection for humanoid push recovery and classification: deep learning approach,” Neural Computing and Applications, vol. 28, no. 3, pp. 565-574, 2017.

[53] V. B. Semwal, M. Raj, and G. C. Nandi, “Biometric gait identification based on a multilayer perceptron,” Robotics and Autonomous Systems vol. 65, pp. 65-75, 2015.

[54] V. B. Semwal, N. Gaud, and G. C. Nandi, “Human gait state prediction using cellular automata and classification using ELM,” in machine intelligence and signal analysis, M.Tanveer and R. B. Pachori, Eds. Singapore: Springer, 2019, pp. 135-145.

[55] D. Gupta, and R. Rani, “Big Data Framework for Zero-Day Malware Detection,” Cybernetics and Systems, vol. 49, no. 2, pp. 103-121, 2018.

[56] Scikit-Learn Machine Learning in Python. Accessed: June. 2019. [Online]. Available: https://scikit-learn.org/stable/.

[57] D. Gupta, and R. Rani, “A study of big data evolution and research challenges,” Journal of Information Science, vol. 45, no. 3, pp. 322-340, 2019.

Meghna Dhalaria

Meghna Dhalaria is pursuing Ph.D. in Computer Science and Engineering Department at Jaypee University of Information and Technology, India. She has completed her Master’s degree in Computer Science & Engineering from Thapar Institute of Engineering and Technology, Patiala. Her current research areas include the applications of machine learning and deep learning.

Ekta Gandotra

Ekta Gandotra is currently working as Assistant Professor in the Department of Computer Science and Engineering at Jaypee University of Information Technology, Waknaghat, India. She has around 12 years of teaching and research experience. She has completed her Ph.D. in Computer Science and Engineering from PEC University of Technology, Chandigarh, India. Her research areas include network & cyber security, malware threat profiling, cyber threat intelligence, machine learning and big data analytics.