Hindawi
Security and Communication Networks
Volume 2020, Article ID 8861639, 14 pages
https://doi.org/10.1155/2020/8861639

Research Article

cHybriDroid: A Machine Learning-Based Hybrid Technique for Securing the Edge Computing

Afifa Maryam,[1] Usman Ahmed,[2] Muhammad Aleem,[3] Jerry Chun-Wei Lin,[2] Muhammad Arshad Islam,[3] and Muhammad Azhar Iqbal[4]

[1]*Department of Computer Science, Capital University of Science and Technology, Islamabad 44000, Pakistan*
[2]*Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen 5063, Norway*
[3]*National University of Computer and Emerging Sciences, Islamabad 44000, Pakistan*
[4]*School of Information Science and Technology (SIST), Southwest Jiaotong University, Chengdu 611756, China*

Correspondence should be addressed to Jerry Chun-Wei Lin; jerrylin@ieee.org

Received 13 August 2020; Revised 7 October 2020; Accepted 3 November 2020; Published 27 November 2020

Academic Editor: Gautam Srivastava

Smart phones are an integral component of the mobile edge computing (MEC) framework. Securing the data stored on mobile devices is very crucial for ensuring the smooth operations of cloud services. A growing number of malicious Android applications demand an in-depth investigation to dissect their malicious intent to design effective malware detection techniques. The contemporary state-of-the-art model suggests that hybrid features based on machine learning (ML) techniques could play a significant role in android malware detection. The selection of application's features plays a very crucial role to capture the appropriate behavioural patterns of malware instances for a useful classification of mobile applications. In this study, we propose a novel hybrid approach to detect android malware, wherein static features in conjunction with dynamic features of smart phone applications are employed. We collect these hybrid features using permissions, intents, and run-time features (such as information leakage, cryptography's exploitation, and network manipulations) to analyse the effectiveness of the employed techniques for malware detection. We conduct experiments using over 5,000 real-world applications. The outcomes of the study reveal that the proposed set of features has successfully detected malware threats with 97% F-measure results.

1. Introduction

Internet of things (IoT), along with edge computing, has revolutionized industrial processes with the help of mobile devices such as tablets, smartphones, smartwatches, and PDAs. Nowadays, mobile devices can adequately render advanced functionalities for efficient, reliable, and scalable cloud services that exploit mobile edge computing (MEC). Extensive usage of Android mobile devices attracts the number of malwares to do MEC services. An increasing number of security threats have emerged recently that is used to steal private user information, lead towards bank frauds, and other socioeconomic crimes [1]. To evade the damages caused by such threats, different malware detection systems [2–4] were presented. Android security solutions for vulnerability assessment and malware analysis can be divided into two main categories as: (1) static and (2) dynamic analysis approaches. In the static technique, the application code is analysed without executing it. The dynamic technique focuses on analysing applications during execution and monitors its interaction with the other system modules and networks [5–7]. However, majority of the existing malware analysis techniques do not consider both the permissions and intents to analyse Android malware.

In contrast to static analysis, most of the dynamic techniques [8, 9] only focus on analysing system and API calls. Existing dynamic malware analysis techniques do not focus on important dynamic features, such as data leakages, network connection manipulation, and enforcing special permissions. Using multiple dynamic features could strengthen the run-time analysis to detect a variety of malicious activities and application security threats. A

comprehensive dynamic approach can detect most of the vulnerabilities and security threats at the cost of execution overhead. To efficiently cope with these issues, there should be a comprehensive malware analysis approach that exploits the lightweight static analysis for the already known malware and a comprehensive dynamic approach for the analysis of zero-day malware threats. In this work, we propose a comprehensive framework that incorporates both the static and dynamic analysis exploiting permissions and intents and considers important dynamic features such as data leakages, network connection manipulation, and enforcing special permissions. The major contributions of this research include the following:

(1) a novel machine learning-based framework to analyse Android applications using a hierarchical approach (applying both the static and dynamic analysis) to detect known and zero-day malware,

(2) a machine learning-based comprehensive static analysis model that incorporates both the application's permissions and intents,

(3) a dynamic analysis model that involves the investigation of system calls (such as network activity, files access, SMS activity, and call activity), external DexClass usage, cryptographic activity, run-time permissions enforcement, and rehashing to detect known and zero-day malware,

(4) hyper-tuning malware classifiers using the tree-based pipeline optimization technique to improve the accuracy for malware detection.

2. Literature Review

This section encompasses the critical analysis of existing state-of-the-art approaches related to malware analysis as shown in Table 1.

2.1. Malware Detection Using Static Analysis. Arora et al. [22] suggested a static approach to analyse permissions using the manifest file. A lightweight technique for malware detection was proposed, and its effectiveness was experimentally demonstrated using real Android malware samples. It extracted the permissions from the manifest file and compared them with a predefined keyword list. The designed model considered only one aspect of vulnerability but ignored other aspects, for example, intents and API calls, among others. Another study [10] considered intents (both the explicit and implicit) as semantically rich features to encode the malicious intentions of malware, especially when the intents are used in combination with permissions. The proposed system performed encoding and extracted explicit and implicit intents, intent filters, and permissions. Almin and Chatterjee [5] utilized the k-means clustering algorithm to classify applications which exploited the permission authorization to do malicious activity. The comparison of the research with famous antivirus solutions indicated that the proposed technique was able to detect the malware that remains undetected by most of the antivirus software. MalDozer [18] is a system relying on artificial neural network that took an input of the raw sequences of API method calls with the same order as they showed up in the .dex file for android malware detection and their family recognition. During the training, MalDozer can automatically recognize malicious patterns using only the sequences of raw method calls in the assembly code. A framework [17] using several features that reflects multidimensional characteristics of the Android applications useful for malware detection is proposed. The authors choose a multimodal deep neural network to select the features with different characteristics. They focused on static features such as Opcode, API, permissions, component, and environmental and string features. Experiments were conducted using the data set from VirusShare and Malgenome project. The proposed system attained a good accuracy of up to 98%. Though they studied many static features, the authors use dynamic features useful to detect zero-day and obfuscated malware. Wang et al. [16] recommended a deep learning-based hybrid model using autoencoder (i.e., DAE) and convolutional neural network (CNN) to improve the accuracy of malware detection. Reconstruction of the multiple features of android application performed and multiple CNN were employed for effective malware detection. To boost feature extraction proficiency, several pretraining procedures were accomplished, and customized combination of the deep autoencoder and CNN model (i.e., DAE-CNN) was employed that various learned ranges of patterns in a short time. The empirical test was performed on a data set comprises 23,000 Android applications with the attained 99.8% accuracy.

2.2. Malware Detection Using Dynamic Analysis. The dynamic analysis technique is based on observing the application behavior during execution. In 2012, Google introduced a dynamic analysis-based security infrastructure named Bouncer by Wang et al. [16] for the Android platform. According to Google officials [16], every application that is uploaded on the Google play store is first simulated on Google Cloud infrastructure (using software named Bouncer). The Bouncer aims at guarding the Google Play store against malware threats.

Canfora et al. [8] introduced a detection method to identify malware attacks by employing system calls. Authors assumed that malicious behaviors were implemented by a sequence of system calls. The study employed a machine learning classifier SVM [23] to identify the specific sequence of system calls associated with malware. Authors used the sequence to identify the new malware families. Though the results of this research work produced a promising accuracy of up to 97%, more features like API calls and network statistics should be explored for a comprehensive dynamic analysis and a higher detection rate. A technique, named IntelliDroid is introduced by Wong and Lie [4], to capture the malicious activities during run time of an application. The IntelliDroid recorded instances of specific API calls. The inputs generated by the proposed system triggered different events to monitor application behavior. In [11], the authors suggested an API sequence analysis-based dynamic

Table 1: A summary of related work.

References	Methodology				Used feature		
	Static	Dynamic	Hybrid	ML-based	Static	Dynamic	Data set
Feizollah et al. [10]	✗	✗	✗	✗	Permission	✓	Custom | Drebin
Almin et al. [5]	✓	✗	✗	✗	Intents	✓	Custom
Canfora et al. [8]	✗	✓	✗	✓	✗	System calls	Custom | Drebin
Wong et al. [4]	✗	✓	✗	✗	✗	Malware tracking through input Genera-	Custom | Drebin
Youngjoon et al. [11]	✗	✓	✗	✗	✗	API calls	Custom
Alzaylaee et al. [2]	✗	✓	✗	✗	✗	API calls	Malgenome data set
Zhao et al. [12]	✗	✓	✓	✓	Permissions	General dynamic activities trigged by	Custom
Dash et al. [13]	✓	✓	✗	✓	✗	System calls, Decoded binder communication, abstracted behavioural patterns	Custom
Xu et al. [14]	✗	✓	✓	✓	Collect attack tree path	Graph kernels	Custom
Yuan et al. [15]	✓	✓	✓	✓	Permissions, sensitive API	DexClass, receive net service start	Custom
Wang et al. [16]	✓	✗	✗	✓	Permissions, API calls, hardware features, code patterns	✗	Custom
Kim et al. [17]	✓	✗	✗	✓	Opcode, API, permissions, component, and environmental and string features	✗	Custom | malgenome data set
Karbab et al. [18]	✓	✓	✗	✓	API method calls	✗	Drebin | malgenome | virushare | contagio minidump
Arshad S et al. [19]	✓	✓	✓	✓	Hardware components requested per missions, application components, and API calls.	System calls	Drebin
Hou et al. [20]	✗	✓	✗	✓	Linux kernel system calls	✗	Custom
Pektas and Acarman [21]	✓	✓	✗	✓	Permissions and hidden payload	API calls, installed services, network connections	Virushare

mechanism. To monitor a new program, the hooking process (part of the implemented tool) monitors and tracks the API call sequences of programs. After extracting the API call sequences, the proposed system is compared with the API call sequence reference database. If matched, an alert about the potential malware is generated.

Alzaylaee et al. [2] proposed a system named DynaLog to extract many features (such as logging of high-level behavior and API calls). The extracted features were further analysed to detect malicious applications. The DynaLog took advantage of existing open-source tools such as Droidbox [24] that can detect a wide range of Android malware. The DynaLog is basically based on the Monkey tool [25] provided by Google for testing Android applications. The applications which were unable to run in the emulated environment remain unchecked by the proposed system. Moreover, the DynaLog was incapable of recording events from the native code within Android applications.

2.3. Hybrid Malware Analysis Techniques. A hybrid malware analysis technique combines the features from both the static and dynamic approaches to detect the wide range of Android security threats. Zhao et al. [12] proposed a hybrid malware analysis technique named AMDetector that employs a modified attack tree model [26] for malware analysis. The static part of the proposed technique detects possible attacks and employs this knowledge to classify applications into benign and malware classes. The application behavior triggered by different code components during run time is the part of the proposed dynamic analysis. The organized rules (with attack trees) rendered the good code coverage to the prototype model. The major drawback of the proposed system was the manual formation of rules and time-costly dynamic analysis.

Bläsing et al. [27] suggested the Application Sandbox (Sandbox) system, which is capable to identify malicious applications using the hybrid analysis. The static part of the

proposed analyzer extracted the classes (i.e., .dex files) and decompiled these files into human-readable format. Furthermore, the code is scanned for suspicious patterns. The proposed system recorded the low-level details of system interactions during the application execution within the sandbox environment. The sandbox environment ensured the security of analysing system and safety of data of the underobservation device. The dynamic part of the proposed technique employs the Monkey tool [25] to observe the behavior of an application by producing random events. One of the limitations observed within the system was its incapability to detect unknown or new types of malware.

SAMADroid [19] represents a hybrid malware detection model that combined the benefits of three different levels: (1) static and dynamic analysis; (2) host, which is local and remote, and (3) machine learning. Static analysis was performed on remote host considering the features belong to hardware components, requested permissions, application components, and API calls. The dynamic analysis was performed on local host using system calls that helped in the detection of malware patterns. Experimental results show that SAMADroid achieves up to 98% malware detection accuracy. Thus, the inspection of the applications is statistical. However, the employed dynamic analysis is only for the system calls related analysis. The employed dynamic analysis of system calls is already a well-worked area [10], and many malwares easily bypass system calls inspections [28] using code obfuscation techniques. Therefore, there is a need to check the other dynamic features like network activity, API calls, and executable codes.

A technique to discover all the flow paths of most engaging APIs in a program using static analysis was proposed [29]. They preferred static analysis because dynamic analysis is sometimes unable to extract all the important APIs completely. This technique is then named DroidDomTree. The strategy that they opted dependent on the study of dominant API is called during static analysis of an application. These dominant API calls are also known as (semantic signatures), and mining the dominance tree of these semantic signatures is used to detect malware. Furthermore, in the dominance tree, authors assigned weights to individual nodes for effective feature selection. This weighting arrangement supported to choose imperative modules that helped further in feature selection and malware detection. The DroidDomTree detection rate ranged between 98.1% and 99.3%. This study proposed the DL-Droid, a dynamic analysis-based Android malware detection scheme by using deep learning to find malicious patterns in a specific application. Authors enhanced their techniques through a state-based input generation method for improved code coverage. DL-Droid examined the accomplishment of the stateful input generation method using random input generation as a relational baseline. They obtained higher accuracies with these stateful approaches. This study highlighted the significance of enhanced input generation for Android malware detection systems during dynamic analysis. The authors conducted experiments using real devices and achieved a detection rate of 97.8% with dynamic features [24].

Chaulagain et al. [30] suggested a deep learning-based hybrid classifier for the safety screening of Android-based applications. The proposed approach takes advantages of automated feature engineering and the combines benefits of static and dynamic analysis. This research collects different artifacts during static and dynamic analysis and trains the deep learner to get independent models. These separate models combined to create a hybrid classifier that helped in vetting decision. The suggested vetting system has proved efficient against imbalance data and has achieved 99% accuracy. Pektas and Acarman [21] presented a hybrid feature-based classification system that statically analysed the requested permissions and the hidden payload while dynamic features such as API calls, installed services, and network connections were considered for malware detection. Different well-known machine learning algorithms were applied to evaluate the accuracy level in the classification of different classifiers using a data set of 3,339 samples. Authors attained the testing accuracy of up to 92% on the employed Android applications. Though the proposed static analysis technique exploits the permissions and payload features, it ignored the close relationship of intents with permissions. Most of the time, considering only permissions to identify the malware is not adequate [10].

Table 1 shows the summary of related work about methodology and important features that most of the researchers employed for static or dynamic analysis. As shown in Table 1, most of the researchers have concatenated either on static or dynamic analysis and ignored an important aspect of application vulnerabilities that can be exploited in both static and dynamic analysis. A few researchers considered hybrid analysis. However, most of them ignored the intents and permissions relationship, which is a crucial aspect of Android applications. Moreover, most of the researchers have not exploited important system calls (such as network activity, file access, SMS, and call activity), usage of external DexClass, data leaks, cryptographic activity, run-time permissions, and rehashing activity during the execution of applications. The critical analysis of narrated state-of-the-art approaches has led us to formulate the following research questions

(i) **Q1:** which of the static features (e.g., permissions along with certain intents patterns) play a vital role in Android malware detection?

(ii) **Q2:** which combination of the dynamic features such as system calls (i.e., network activity, file access, SMS, and call activity), usage of external DexClass, data leaks, cryptographic activity, run-time permissions, and detection of rehashing activity is important for Android malware identification?

(iii) **Q3:** how can malware detection rate be improved by employing hybrid analysis and machine learning-based classification?

To address these research questions, we propose a hybrid machine learning-based malware detection framework called *HybriDroid* for Android platform.

3. Proposed Hybrid Malware Analysis

To analyse the impact of hybridization, we propose two machine learning-based hybrid malware analyzers, respectively, named *HybriDroid* and ***cHybriDroid***. The *HybriDroid* framework exploits static as well as dynamic features for malware analysis using a hierarchical mechanism. First, the applications are analysed solely using the static features, and then the dynamic features are employed to examine the suspicious (the applications marked as clean by the static analysis) applications. Moreover, to investigate the impact of combined analysis (using both the static and dynamic features), we propose **cHybriDroid** framework.

3.1. HybriDroid Architecture. This section describes the overall methodology of the proposed Android malware analysis framework, that is, *HybriDroid* (shown in Figure 1). The proposed hybrid approach is comprised of a hierarchical system based on two phases: (1) static and (2) dynamic phases (as depicted in Figure 1). In the static analysis phase, the APK files of applications are first dissembled into XML and Java files. After that, the XML files are examined to extract the application related to permissions and intents.

These features are then supplied to the proposed machine learning-based static analyzer. By employing the provided static features, the machine learning-based analyzer categorizes an application like malware or suspicious. To further examine the suspicious applications, the dynamic analysis phase is initiated. The applications classified as suspicious are then provided to the dynamic analyzer for analysing run-time behaviors.

For dynamic analysis, first of all, each application is executed in the emulated environment (using DroidBox [24] emulation tool) to log the observed dynamic features (such as system calls, usage of external DexClass, data leaks, cryptographic activity, and detection of rehashing activity). The dynamic features are then provided to the machine learning-based dynamic analyzer for the classification purpose. The machine learning-based dynamic analyzer classifies these suspicious applications as benign or malware. The applications classified as malware are added to the malware data set while the applications declared as benign are added to the clean applications data set.

3.2. cHybriDroid Architecture. To investigate the impact of combined analysis (using both the static and dynamic features), we propose a ***cHybriDroid*** framework (as shown in Figure 2). The **cHybriDroid** examines the Android applications using both the static and dynamic features simultaneously (see the architecture of **cHybriDroid** in Figure 2). For each Android application, both the static and dynamic features are extracted and provided to the machine learning-based analyzer for classification (as malware or benign). To extract the static features (i.e., intents and permissions), the application is disassembled into APK and manifest files. Moreover, the application is executed in the virtual environment (interactively by tapping and using sample inputs), and the dynamic features are logged. Afterwards, the static (i.e., intents and permissions) and dynamic (such as data leakage, network usage, and use of DexClass) features are provided for the developed **cHybriDroid** to analyse the application (as depicted in Figure 2).

3.3. Classifier Training for HybriDroid and cHybriDroid. Figure 3(a) depicts the complete training process of the proposed *HybriDroid* malware analyzer. The training data set comprises 50% benign (i.e., clean Android applications) and 50% malware (as mentioned in Table 2). As the *HybriDroid* mechanism is based on the hierarchical model, therefore, both the static and dynamic machine learning analyzers are trained separately. To train the static analyzer, an Android application is disassembled into Java and XML files (sample shown in Figure 4) in order to extract the feature vectors related to permission and intents. The disassembled Java, XML, and manifest files are used to obtain static features such as intents and permissions. These intents and permissions are then compared with each application in the data set. If the application intent or permission matches with the extracted permissions, the value of that intent or permission is set to 1; otherwise, it is set to 0. Similarly, a feature vector based on 407 distinct values is formed. These feature vectors along the application category or label (i.e., malware or benign) are provided to the static machine learning analyzer. Similarly, for the training of the dynamic analyzer (in the *HybriDroid* framework), 50% of the benign and 50% of the malware applications-based training data set was executed in a virtual environment (i.e., DroidBox [24]). A total of 15 distinct dynamic features are collected and provided along with the application category (i.e., malware or benign) to the dynamic analyzer (*HybriDroid*). Additionally, *K-fold cross-validation* method is used along with grid search mechanism that is employed for hyperparameter tuning (as shown in Table 3).

Figure 3(b) shows the training of **cHybriDroid** that employs single machine learning-based analyzer trained using both the static and dynamic features simultaneously. For each Android application, the static and the dynamic features are extracted and supplied along with the application category (i.e., benign or malware) to the **cHybriDroid**'s combined analyzer. The combined analyzer is trained using 432 distinct feature vectors based on the static and dynamic aspects of the application.

4. Experimental Result

The experiments are performed on a personal computer. Detailed specifications of the machine are illustrated in Table 4. To evaluate the proposed frameworks, *HybriDroid* and **cHybriDroid**, we employed five machine learning classifiers, respectively, are *Random Forest* (RF), *K star* (K∗), *Naive Bayes* (NB), *Support Vector Machine* (SVM), and J48 decision tree [12–14, 27, 31]. Moreover, TPOT [28] technique is also used that chooses the right machine learning model and the best hyperparameter for that model.

4.1. Data Set. The benign or clean applications in the data set are collected from the Google play store [16], and a third-party app store called Apkpure [16] is shown in Table 5. For

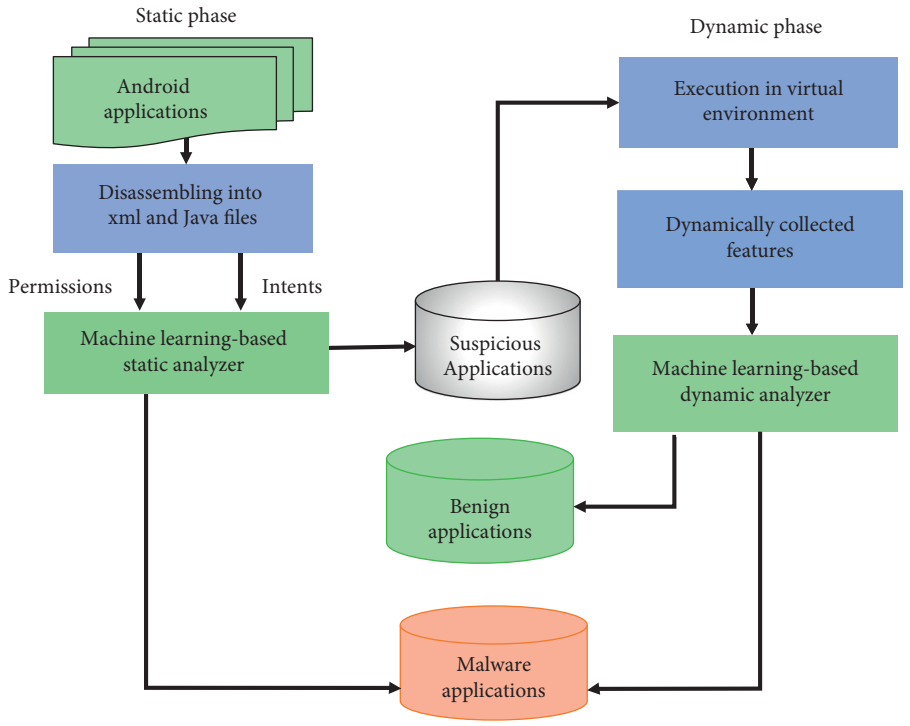

Figure 1: Architecture of *HybriDroid*.

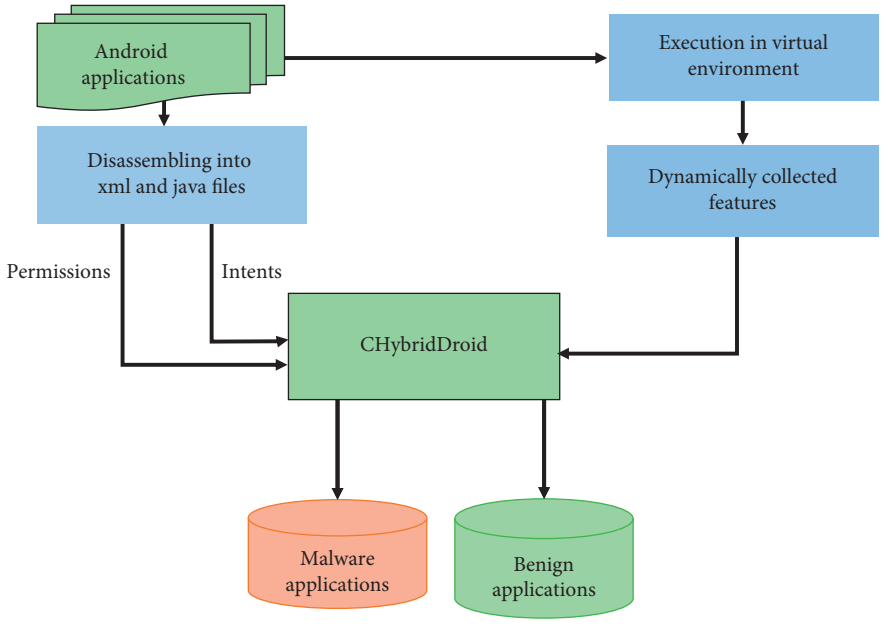

Figure 2: Architecture of cHybriDroid.

malware samples, we acquired benchmark Drebin [3] data set that consists of 5,560 malwares from 179 different families and some of them are shown in Table 6. Drebin is extensively used throughout research works on Android malware detection. The Drebin data set consists of malware applications obtained from various Android markets, different antivirus engines, malware forums, security blogs, and Android *malgenome* project [16].

4.2. Feature Selection. The permissions are one of the important static features which must be examined carefully to safeguard from the potential security threats. In addition to the permissions, intents within Android applications are another important aspect requiring careful analysis. Intents are part of the complex messaging model of Android system, which facilitates execution of the different applications, services, and operating system functions. Different activities,

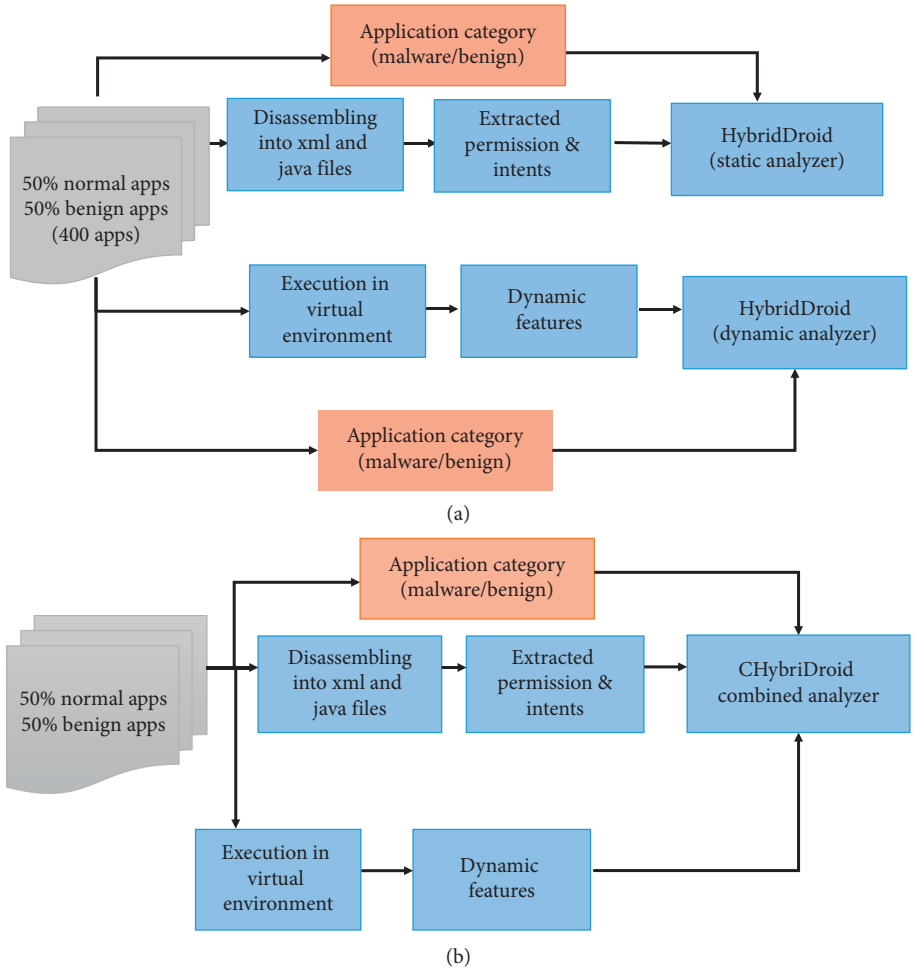

Figure 3: Training methodology. (a) Training of static and dynamic machine learning analyzers for *HybriDroid*. (b) Training of the combined analyzer for cHybriDroid.

Table 2: Data set details.

Application type	Number of applications	Applications categories
Benign	2500	28 different categories
Malware	2500	178 different families

broadcast receivers, and some services used intents for their activation and record their type of intent using intent filters in the manifest file. Some of the recent studies [10, 32] have shown that the intents and permissions are often exploited (such as intent spoofing and permission collusion) by the malware. Thus, their critical examination is necessary to detect malicious activities. Table 7 shows the features collected (using DroidBox tool) during the dynamic analysis step of the proposed methodology. These features are the result of the execution events generated during the execution of applications (within a virtual environment). From Table 8, it is evident that the internet is the most employed (i.e., 20%) permission by the applications (by both the malware and benign). Other permissions that are the part of the most requested permission set in malware applications belong to sending and writing SMS, having a collective percentage of 14. Moreover, accessing approximate and exact locations through *ACCESS_FINE_LOCATION* and *ACCESS_COARSE_LOCATION* permissions is employed by the 11% malware applications.

4.3. Feature Ranking. The motivation behind using a reduced feature set (for the employed predictive models) is to eliminate redundant data, reduce overfitting issues, improve classification accuracy, and decrease the training time of the algorithm. The dynamic analysis results in a large number of features; therefore, it was necessary to use only the important features for the machine learning model. For this purpose, we employ the information gain method [16] that finds certain patterns of the features in the employed applications of the data set. Each feature is assigned with a certain score highlighting the effectiveness of the feature in classification. The *InfoGain* is a well-known feature selection algorithm that records the changes in the entropy of the information class before and after the observation [3]. The formula to measure the information gain is shown as

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.google.samples.dataprivacy">

    <uses-feature
        android:name="android.hardware.camera"
        android:required="true" />

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme"
        android:fullBackupContent="@xml/backup_descriptor"
        tools:ignore="GoogleAppIndexingWarning">



        <activity android:name=".page.images.ImagesActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
```

Figure 4: Sample of the manifest .xml file.

Table 3: Grid search setting.

Classifier	Candidate	Parameters
Decision tree	462	{'max_features': ['auto'-'sqrt'-'log2']-'min_samples_split': [2-3-4-5-6-7-8-9-10-11-12-13-14-15]-'min_samples_leaf': [1-2-3-4-5-6-7-8-9-10-11]-'random_state': [123]}
Random forest	288	{'Bootstrap': [True]-'max_depth': [80-90-100-110]-'max_features': [2-3]-'min_samples_leaf': [3-4-5]-'min_samples_split': [8-10-12]-'n_estimators':[100-200-300–1000]}
SVM	14	{'C': [6-7-8-9-10-11-12]- 'kernel': ['linear'-'rbf']}
Kstar	192	{'n_neighbors': [5-6-7-8-9-10]-'leaf_size':[1-2-3-5]-'weights':['uniform'-'distance']-'algorithm':['auto'-'ball_tree'-'kd_tree'-'brute']-'n_jobs':[−1]}
Naive Bayes	—	—
Tpot	Generative model	—

Table 4: Experimental setup.

Processor	Intel core (TM) i7-4720HQ 2.60 GHz
Memory	16 GB
Operating system	Ubuntu 16.04LTS
Machine learning tool	Weka 3.6

$$\text{infoGain}(P, F) = \text{Entropy}(P) - \sum_{v \in V(F)} \frac{|P_v|}{|P|} \cdot \text{Entropy}(P_v), \tag{1}$$

where P indicates the set representing the pattern, $|P|$ is the number of samples in P, v is the value of the feature F, (P, v) is the value of feature F, and Pv is the subset of P (where feature F has value v). Before the observation of features entropy, the class is defined and shown as

$$\text{Entropy}(P) = \sum_{c \in C} \frac{|P_c|}{|P|} \cdot \log_2 \frac{|P_c|}{|P|}, \tag{2}$$

where C indicates the class set and Pc represents the subset of P belonging to class c. Information gain is considered as a simple and fast ranking method that yields the most suitable features, which are helpful in identifying application class (in our case malware or benign). Using InfoGain, 172 important static features (comprising permissions and intents) out of a total of 407 features are selected. The top 10 features are

Table 5: Nonmalware application details.

S.NO.	Applications categories
1	Health & fitness
2	Art & design
3	Beauty
4	Business
5	Communication
6	Education
7	Event
8	House & home
9	Sports
10	Productivity
11	Photography
12	Camera
13	Finance
14	Auto & vehicles
15	Travel and local
16	Food & drink
17	Lifestyle
18	Video players & editors
19	Weather
20	Social
21	Shopping
22	Tools
23	Parenting
24	News & magazines
25	Music & audio
26	Medical
27	Entertainment
28	Music & audio

Table 6: Malware application details.

S.NO.	Malware family
1	Plankton
2	DroidKungFu
3	GinMaster
4	FakeDoc
5	FakeInstaller
6	Opfake
7	BaseBridge
8	Nisev
9	Adrd
10	$K_{\min}$
11	Geinimi
12	DroidDream
13	FakeRun
14	Iconosys
15	SmsWatcher
16	UpdtKiller
17	Gappusin
18	Proreso
19	Mobsquz
20	Cosha
21	SpyMob
22	Coogos
23	Updtbot
24	Ackposts
25	Fatakr
26	Vidro
27	Booster
28	EWalls

ranked and shown in Table 9. These results show that *Send_SMS* and *Receive_SMS* static features have attained the highest rank value compared with the other static features.

For intents, the *receiver* has been ranked highest among the intent category. The top ranked dynamic features with the rank score are shown in Table 10. As shown in Table 10, *sendsms* is the top dynamic feature that has the highest potential to reveal the category of an Android application (i.e., as malware or benign). *Sendsms* dynamic feature represents information leakage via network, SMS, or any file-based activity. *Cryptousage*, *sendsms*, *enfperm*, and *sendnet* are the other top-ranked features which retain maximum information (i.e., attained higher rank value), and this shows the significance of these features for malware analysis. In this research, we use the top five (out of a total of 15) ranked dynamic features.

The *dataleaks* dynamic feature retains the maximum information when *InfoGain* is applied (as shown in Table 10). This information is necessary for accurate malware classification. Similarly, the *READ_SMS* from the intent category has the highest potential to accurately classify malware compared with the other employed features. The *android.provider.Telephony.SMS_RECEIVED* in the permission's category is among the top 10 highest-ranked permission (as shown in Table 11). In this research, we selected the top 20 (422) hybrid features. The full feature ranking and information gain are mentioned at https://bit.ly/2GduUEt.

4.4. Result Discussion. In Table 12, results related to cross-validation grid search experiment are presented. When feature selection is not performed then, TPOT produces the highest 0.91 F-measure. However, the Naive Bayes produces 0.98 precision, and TPOT produces 0.91 recall. In Table 12, when feature selection is performed, the TPOT F-measure is decreased from 0.91 to 0.87. Random forest produced the best result of 0.88 F-measure and 0.88 precision. The reduced result of the model indicated that removed features have minimal impact on the performance of the classifiers. In Table 12, the cross-validation grid search experiment related data based on dynamic features with and without feature selection is presented. When feature selection is not performed then, TPOT produces the highest 0.94 F-measure, which is 0.03% improved compared with the static features mentioned in Table 13. However, the Naive Bayes produces 0.99 precision and support vector machine produces 0.92 recall. In Table 12, when feature selection is employed, the TPOT F-measure is decreased from 0.94 to 0.91. The TPOT produced the best result of 0.91 F-measure and 0.88 precision while reducing the number of features from 15 to 5 (with a drop of F-measure 0.03). Table 13 shows the best classifier for dynamic features based analysis. In Table 12, cross-validation grid search experiment is conducted on the hybrid features with (20 selected features) and without (total 422 features) feature selection. When feature selection is not performed then the Naive Bayes produces the highest F-measure (i.e., 0.99) which is

Table 7: Applications features for dynamic analysis.

No.	Feature	Description
1	DexClass	Actions of loading external dex function
2	Opennet	Facilitate the connection for network
3	Service start	Log the services in operation
4	Close net	Close network connection
5	Send net	Data transmit/sent to the network
6	Recvnet	Data received from the network
7	Data leaks	Detect leakage of information on the phone including messages, e-mail, password, contacts, IMEI, GPS information phone number, and so on
8	Accessed files	File accesses
9	Fda ccess	Read and write operations of file and directory
10	Send sms	Send SMS
11	Phone call	Phone calls made
12	Cryptousage	Detect the cryptographic functions and what key is used when encrypting and decrypting data
13	Recvaction	APKs function invoked as a receiver
14	Enfperm	Enforce special permission to activity, broadcast receiver, and service
15	Hashes	The hash value of APK file

Table 8: Most used (percentage of occurrence) permissions and intents in the applications.

Frequent permissions	Percentage	Frequently asked intents	Percentage (%)
Internet	20	Action.main	28
Read_phone_state	15	Category.launcher	24
Access_ network_state	13	Boot_completed	14
Write_external_storage	7	Category.default	8
Write_sms	5	Sms_received	8
Send_sms	9	Phone_state	5
Receive_boot_completed	11	Category_home	4
Wake_lock	9	New_outgoing_call	3
Access_fine_location	6	ACTION.VIEW	3
Access_coarse_location	5	Category.browsable	3

Table 9: Static features (ranked using info gain).

Ranked	Importance	Features name
1	0.18147	SEND_SMS
2	0.16383	com.Google.android.c2dm.intent.RECEIVE
3	0.16296	com.Google.android.c2dm.permission.RECEIVE
4	0.14353	com.android.vending.INSTALL_REFERRER
5	0.13254	READ_PHONE_STATE0
6	0.12882	0com.Google.firebase.INSTANCE_ID_EVENT
7	0.12363	READ_EXTERNAL_STORAGE
8	0.12227	ACCESS_NETWORK_STATE
9	0.12148	c1dm.intent.REGISTRATION
10	0.11792	C2D_MESSAGE
11	0.10674	category.BROWSABLE
12	0.10086	android.intent.action.VIEW
13	0.09721	RECEIVE_SMS
14	0.08396	READ_SMS
15	0.07374	GET_ACCOUNTS
16	0.07059	Telephony.SMS_RECEIVED
17	0.06276	GET_TASKS
18	0.06225	com.android.vending.BILLING
19	0.0551	android.intent.action.SEND
20	0.0546	READ_LOGS/writeLogs

0.05% improved compared with dynamic features and 0.08% improved result compared with the static features mentioned in Table 13, respectively. The Naive Bayes produces the precision of 1.00 and the recall of 0.99. In Table 14, when feature selection is performed, the TPOT F-measure results in 0.97 and the Naive Bayes F-measure is decreased from 0.99 to 0.96. The TPOT produced the best results, that is, 0.97 F-measure, 1.00 precision, and

Table 10: Dynamic features (obtained using InfoGain).

Rank	Importance	Feature name
1	0.2654	Sendsms
2	0.2425	Dataleaks
3	0.1872	Cryptousage
4	0.0913	Enfperm
5	0	Accessedfiles
6	0	Servicestart
7	0	Recvnet
8	0	Sendnet
9	0	Phonecalls
10	0	Closenet
11	0	Opennet
12	0	Hashes
13	0	DexClass
14	0	Recvsaction
15	0	Fdaccess

0.94 recall, while reducing the features from 422 to 20, with a drop of F-measure up to 0.02.

Table 13 showed each fold result of the TPOT (without feature selection) and random forest (with feature selection). Since random forest is trained on different samples of the data which reduces variance, it obtained better performance. Moreover, random forest used a random subset of features which also helps to reduce overfitting. The dynamic features based TPOT technique is shown in Table 13 depicting the most performing classifiers (with and without employing feature selection). The reason that the **extratree** classifier obtained the improved results compared with the other classifier is that the random value is selected for feature consideration. The random split for the extra trees helps to create more diversified trees and less *splitters*. Table 13 shows the best classifier using the hybrid features. In the hybrid feature, Naive Bayes classifier resulted in the best classifier without feature selection and the TPOT-based technique results in best classifier with reduced features. The Naive Bayes is a probabilistic based classifier, so it does not require any selection of tune parameter. However, TPOT needed hypertuning, where we used the evolutionary algorithm to optimize the parameter. The tune parameters for TPOT model are *StackingEstimator (estimator = LogisticRegression (C = 0.1, dual = True, penalty = "l2")), GaussianNB ()).* The reason that TPOT technique obtained the improved results compared with the other classifiers is that it uses a stack generation technique to improve its performance. The metalearner that outputs Gaussian classifier makes the final prediction. The results presented in Table 13 show that the TPOT and Naive Bayes outperformed the other machine learning models and are more effective in malware detection. The attained F-measure value for the TPOT model indicates the notable performance of the model. It is evident that, for the TPOT model, the true positive rate is observed fairly high and the false positive rate is extremely low. Therefore, we employ the TPOT classification technique for our proposed **cHybriDroid** framework.

4.5. Prediction Model Overhead. The **cHybriDroid** is trained offline. The overhead of using **cHybriDroid** predictor includes the selective feature extraction and making the predictions. The overhead of feature extraction is negligible (approximated 1s in total) as a feature is extracted at compile time. The prediction model training is performed once, and it is a one-time cost. The training and testing time for both models are mentioned in Table 15. In summary, the overhead of the prediction model is negligible, that is, two seconds for one application.

Table 11: Hybrid features (information gain).

Rank	Importance	Features name
1	0.364	Dataleaks
2	0.312	SEND_SMS
3	0.305	Sendsms
4	0.298	Servicestart
5	0.294	Opennet
6	0.285	READ_SMS
7	0.285	RECEIVE_SMS
8	0.272	ACCESS_COARSE_LOCATION
9	0.272	android.provider.Telephony.SMS_RECEIVED
10	0.259	Sendnet
11	0.259	Recvnet
12	0.252	ACCESS_FINE_LOCATION
13	0.249	Cryptousage
14	0.248	READ_PHONE_STATE0
15	0.23	ACCESS_NETWORK_STATE
16	0.226	WRITE_SMS
17	0.22	Enfperm
18	0.22	CAMERA
19	0.217	action.BOOT_COMPLETED
20	0.214	Internet

Using the hybrid analysis approach, we experimented with real malware and benign Android applications. Our study showed that using both the static and dynamic application features result in a commendable malware detection accuracy. With the feature ranking mechanism, we further optimized the two proposed hybrid methodologies in terms of performance and accuracy. The reduced number and employing only the important features results in good detection performance and accuracy. For hybrid malware analysis, we adopted two strategies: (1) *HybriDroid* and (2) **cHybriDroid**. The *HybriDroid* methodology was typically designed to perform a hybrid malware analysis (employing both the static and dynamic or run-time features) using a hierarchical mechanism. At the same time, the **cHybriDroid** mechanism was employed to analyse the effectiveness of malware detection when the static and dynamic features are analysed simultaneously. Our results exhibit a higher malware detection accuracy for the *HybriDroid* with a 97% F-measure as mentioned in Table 16. We found that the TPOT [28] was the top-performing machine learning model (for **cHybriDroid**) as compared with the other employed models. To attain a better performance insight, we noted the *False Positive Rate* (FPR) and *True Positive Rate* (TPR) for the **cHybriDroid** classifier. The results revealed that the TPOT [28] machine learning model attained the highest performance up to 96% TPR. Similarly, the r^2 value for the TPOT machine learning model also specifies the potential of TPOT to detect malware. Overall, the malware detection accuracy of the hierarchical hybrid approach (i.e.,

Table 12: The summary of results with and without feature selection.

	(a) Static features results			(b) Dynamic features results			(c) Hybrid feature results		
	F-meas.	Prec.	Recall	F-meas.	Prec.	Recall	F-meas.	Prec.	Recall
WO Feat. Sel.									
SVM	0.90	0.90	0.91	0.90	0.90	**0.92**	0.93	1.00	0.89
Decision tree	0.83	0.84	0.84	0.84	0.85	0.84	0.88	0.96	0.83
Random forest	0.89	0.85	0.83	0.89	0.86	0.84	0.96	0.98	0.93
K-star	0.84	0.91	0.8	0.85	0.92	0.80	0.57	1.00	0.42
Naive Bayes	0.85	**0.98**	0.75	0.85	**0.99**	0.75	**0.99**	**1.00**	**0.99**
TPOT	**0.91**	0.92	**0.91**	**0.94**	0.98	0.90	0.99	1.00	0.99
With feat. sel.									
SVM	0.86	0.86	0.88	0.90	0.94	0.87	0.95	1.00	0.91
Decision tree	0.87	0.86	0.89	0.91	0.95	0.87	0.91	0.97	0.87
Random forest	**0.88**	**0.88**	0.89	0.90	0.94	0.87	0.95	0.96	0.95
K-star	0.84	0.77	**0.94**	0.83	0.83	0.87	0.84	1.00	0.74
Naive Bayes	0.79	0.68	**0.94**	0.83	**0.97**	0.73	0.96	1.00	0.93
TPOT	0.87	0.87	0.88	**0.91**	0.94	**0.89**	**0.97**	**1.00**	**0.94**

Table 13: The selected best model in terms of cross-validation score.

	$k=1$	$k=2$	$k=3$	$k=4$	$k=5$	$k=6$	$k=7$	$k=8$	$k=9$	$k=10$	Average
(a) Static features											
TPOT, without feature selection											
F-measure	0.92	0.82	0.82	0.97	0.93	0.86	0.97	0.92	1.00	0.93	0.91
Precision	1.00	0.75	0.75	1.00	0.90	1.00	1.00	0.95	1.00	0.90	0.93
Recall	0.85	0.90	0.90	0.95	0.95	0.75	0.95	0.90	1.00	0.95	0.91
Random forest, with feature selection											
F-measure	0.92	0.74	0.86	0.93	0.85	0.83	0.97	0.93	0.97	0.84	0.89
Precision	0.95	0.65	0.79	0.90	0.84	0.93	1.00	0.90	0.95	0.89	0.88
Recall	0.90	0.85	0.95	0.95	0.85	0.75	0.95	0.95	0.95	0.80	0.89
(b) Dynamic features											
TPOT, without feature selection											
F-measure	0.90	0.90	0.97	0.85	0.79	0.97	1.00	1.00	1.00	1.00	0.94
Precision	1.00	0.93	1.00	0.91	1.00	1.00	1.00	1.00	1.00	1.00	0.98
Recall	0.87	0.93	0.73	0.67	0.80	1.00	1.00	1.00	1.00	1.00	0.90
TPOT, with feature selection											
F-measure	0.89	0.90	0.93	0.67	0.84	0.93	0.97	1.00	1.00	1.00	0.91
Precision	1.00	0.93	1.00	0.75	0.81	0.93	1.00	1.00	1.00	1.00	0.94
Recall	0.80	0.87	0.87	0.60	0.87	0.93	0.93	1.00	1.00	1.00	0.89
(c) Hybrid features											
Native Bayes, with feature selection											
F-measure	0.93	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
Precision	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Recall	0.88	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
TPOT, with feature selection											
F-measure	0.93	1.00	1.00	0.92	0.92	0.92	1.00	1.00	1.00	1.00	0.97
Precision	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Recall	0.88	1.00	1.00	0.86	0.86	0.86	1.00	1.00	1.00	1.00	0.94

Table 14: Research answers.

Q1	Selected 172 out of 407 based on InfoGain method
Q2	Selected 5 out 15 based on InfoGain method
Q3	Hybrid analysis increased the F-measure score of 5% with and without feature selection

Table 15: Training and testing time.

Model	Training time (seconds)	Testing time (seconds)
TPOT	0.09	0.007

Table 16: Hybrid features comparison.

Classifier	TPR	FNR	R^2
TPOT	0.96	0.04	0.91
SVM	0.91	0.09	0.82
Decision tree	0.87	0.13	0.64
Random forest	0.87	0.13	0.73
Kstar	0.65	0.35	0.27
Naive Bayes	0.96	0.04	0.91

cHybriDroid) was marginally better than the combined hybrid approach, that is, *HybriDroid*.

4.6. Analysis. As seen from Table 13, the static, hybrid, and dynamic model achieve the high F-measure score. We train the analysis tool on a comprehensive data set and use the optimized parameters for machine learning. Within the proposed security mechanism, we firstly do the static analysis part mainly comprising the manifest file, including permission tags and application intents. The reason for the static analysis is that malware can be tested on the submission of the application before the execution of the application. If the model probability is low, the mechanism should apply the dynamic classification model and detect it under control environment. The dynamic method takes rigorous testing, so it will cost execution time. If the model is uncertain again, then the proposed method will apply to the hybrid model. In this way, we test the application with three different models. Table 14 answers the research question mentioned in Section 1. The method can be adopted for the ransomware and adversarial attacks. The method can be applied in a huge size data set. We can train such kind of ensemble machine learning analyzer on discussed features to detect the ransomware application and classify them into families.

5. Conclusion and Future Work

Nowadays, Android is deemed as the renowned OS for mobile devices. Subsequently, the Android platform attracts several malware experts to gather huge economic and social benefits. To mitigate malware activities, different malware detection systems have been proposed. However, the deficiencies in these systems have led us to propose a novel machine learning-based hybrid malware detection framework that employs several important static and dynamic features. Furthermore, the study has also analysed the role of different machine learning classifiers for malware detection. This study highlights that, in the development of a robust machine learning-based malware detection system, the selection of features from the data set is one of the significant steps. Feature selection depends upon the analysis method through which they are extracted. It is the analysis technique that determines the compatibility of features with the classification algorithm. In the experiments, we attain 97% F-measure, and the trained classifier shows a tremendous efficiency with an r^2 value of 0.91. The TPR is also high, that is, 0.96, while the FPR is very low, that is, 0.04. For the future work, we intend to incorporate code coverage, memory utilization, and network statistics aspects of the executing applications (for dynamic analysis). Moreover, the classifiers will be trained to subclassify the malware into families.

Data Availability

The datasets used in the study were taken from previously published studies (Google-play store [16]; Drebin [3]).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

The research was partially supported by Western Norway University of Applied Sciences, Norway.

References

[1] P. Calciati, K. Kuznetsov, A. Gorla, and A. Zeller, "Automatically granted permissions in android apps," in *Proceedings of the International Conference on Mining Software Repositories*, pp. 114–124, Montreal, Canada, May 2020.

[2] M. K. Alzaylaee, S. Y. Yerima, and S. S. Dynalog, "An automated dynamic analysis framework for characterizing android applications," 2016, http://arxiv.org/abs/1607.08166.

[3] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," in *Proceedings of the Annual Network and Distributed System Security Symposium*, pp. 1–15, San Diego, CA, USA, March 2014.

[4] M. Y. Wong and D. Lie, "IntelliDroid: a targeted input generator for the dynamic analysis of Android Malware," in *Proceedings of the Annual Network and Distributed System Security Symposium*, University of Toronto, Toronto, Canada, 2016.

[5] S. B. Almin and M. Chatterjee, "A novel approach to detect android malware," *Procedia Computer Science*, vol. 45, pp. 407–417, 2015.

[6] I. A. Dogru and M. Önder, "Appperm analyzer: malware detection system based on android permissions and permission groups," *International Journal of Software Engineering and Knowledge Engineering*, vol. 30, no. 3, pp. 427–450, 2020.

[7] S. Liang and X. Du, "Permission-combination-based scheme for android mobile malware detection," in *Proceedings of the IEEE International Conference on Communications*, pp. 2301–2306, Sydney, NSW, Australia, June 2014.

[8] G. Canfora, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Detecting android malware using sequences of system calls," in *Proceedings of the International Workshop on Software Development Lifecycle for Mobile*, pp. 13–20, Bergamo, Italy, August 2015.

[9] S. J. Hussain, U. Ahmed, H. Liaquat, S. Mir, N. Z. Jhanjhi, and M. Humayun, "IMIAD: intelligent malware identification for android platform," in *Proceedings of the International Conference on Computer and Information Sciences*, pp. 1–6, Sakaka, Saudi Arabia, April 2019.

[10] F. Ali, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "AndroDialysis: analysis of android intent effectiveness in

malware detection," *Computers & Security*, vol. 65, pp. 121–134, 2017.

[11] K. Youngjoon, E. Kim, and H. K. Kim, "A novel approach to detect malware based on API call sequence analysis," *International Journal of Distributed Sensor Networks*, vol. 11, no. 9, p. 659101, 2015.

[12] S. Zhao, X. Li, G. Xu, L. Zhang, and Z. Feng, "Attack tree based android malware detection with hybrid analysis," in *Proceedings of the IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 380–387, Beijing, China, September 2014.

[13] S. K. Dash, G. Suarez-Tangil, S. J. Khan et al., "Classifying android malware based on runtime behavior," in *Proceedings of the IEEE Security and Privacy Workshops*, pp. 252–261, San Diego, CA, USA, May 2016.

[14] L. Xu, P. Z. Dong, M. A. Alvarez, J. A. Morales, X. Ma, and J. Cavazos, "Dynamic android malware classification using graph-based representations," in *Proceedings of the IEEE International Conference on Cyber Security and Cloud Computing*, pp. 220–231, New York, NY, USA, October 2016.

[15] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: deep learning in android malware detection," in *Proceedings of the ACM SIGCOMM Conference*, pp. 371-372, Beijing, China, August 2014.

[16] W. Wang, M. Zhao, and J. Wang, "Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network," *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 8, pp. 3035–3043, 2019.

[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, 2019.

[18] E. B. Karbab, M. Debbabi, A. Derhab, and D. M. Maldozer, "Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, no. S48–S59, 2018.

[19] S. Arshad, M. A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu, "Samadroid: a novel 3-level hybrid malware detection model for android operating system," *IEEE Access*, vol. 6, pp. 4321–4339, 2018.

[20] S. Hou, A. Saas, L. Chen, and Y. Ye, "Deep4maldroid: a deep learning framework for android malware detection based on linux kernel system call graphs," in *Proceedings of the IEEE/WIC/ACM International Conference on Web Intelligence - Workshops*, Omaha, NE, USA, October 2016.

[21] A. Pektas and T. Acarman, "Ensemble machine learning approach for android malware classification using hybrid features," in *Proceedings of the International Conference on Computer Recognition Systems*, Wroclaw, Poland, May 2017.

[22] A. Arora, S. K. Peddoju, and M. C. PermPair, "Android malware detection using permission pairs," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 1968–1982, 2020.

[23] J. Cui, L. Wang, X. Zhao, and H. Zhang, "Towards predictive analysis of android vulnerability using statistical codes and machine learning for iot applications," *Computer Communications*, vol. 155, pp. 125–131, 2020.

[24] M. K. Alzaylaee, S. Y. Yerima, and S. S. Dl-droid, "Deep learning based android malware detection using real devices," *Computers and Security*, vol. 89, 2020.

[25] N. Wongwiwatchai, P. Pongkham, and K. Sripanidkulchai, "Comprehensive detection of vulnerable personal information leaks in android applications," in *Proceedings of the IEEE Conference on Computer Communications Workshops*, Toronto, ON, Canada, July 2020.

[26] S. Valluripally, A. Gulhane, R. Mitra, K. A. Hoque, and C. Prasad, "Attack trees for security and privacy in social virtual reality learning environments," in *Proceedings of the IEEE Annual Consumer Communications & Networking Conference*, Las Vegas, NV, USA, January 2020.

[27] T. Bläsing, L. Batyuk, S. Aubrey-Derrick, S. A. Çamtepe, and A. Sahin, "An android application sandbox system for suspicious software detection," in *Proceedings of the International Conference on Malicious and Unwanted Software*.

[28] R. S. Olson, B. Nathan, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, pp. 485–492, Denver, CO, USA, July 2016.

[29] S. Alam, S. A. Alharbi, and S. Yildirim, "Mining nested flow of dominant APIs for detecting android malware," *Computer Networks*, vol. 167, p. 107026, 2020.

[30] D. Chaulagain, P. Poudel, P. Pathak et al., "Hybrid analysis of android apps for security vetting using deep learning," in *Proceedings of the IEEE Conference on Communications and Network Security*, Avignon, France, June 2020.

[31] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, ""Andromaly": a behavioral malware detection framework for android devices," *Journal of Intelligent Information Systems*, vol. 38, no. 1, pp. 161–190, 2012.

[32] F. Idrees and M. Rajarajan, "Investigating the android intents and permissions for malware detection," in *Proceedings of the IEEE International Conference on Wireless and Mobile Computing*, pp. 354–358, Larnaca, Cyprus, October 2014.