symmetry

MDPI

Article

Permissions-Based Detection of Android Malware Using Machine Learning

Fahad Akbar [1], Mehdi Hussain [1,*], Rafia Mumtaz [1], Qaiser Riaz [1], Ainuddin Wahid Abdul Wahab [2] and Ki-Hyun Jung [3,*]

[1] School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan; fakbar.msis18seecs@seecs.edu.pk (F.A.); rafia.mumtaz@seecs.edu.pk (R.M.); qaiser.riaz@seecs.edu.pk (Q.R.)
[2] Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia; ainuddin@um.edu.my
[3] Department of Cyber Security, Kyungil University, Gyeongsan-si 38428, Korea
* Correspondence: mehdi.hussain@seecs.edu.pk (M.H.); khanny.jung@gmail.com (K.-H.J.)

Abstract: Malware applications (Apps) targeting mobile devices are widespread, and compromise the sensitive and private information stored on the devices. This is due to the asymmetry between informative permissions and irrelevant and redundant permissions for benign Apps. It also depends on the characteristics of the Android platform, such as adopting an open-source policy, supporting unofficial App stores, and the great tolerance for App verification; therefore the Android platform is destined to face such malicious intrusions. In this paper, we propose a permissions-based malware detection system (PerDRaML) that determines the App's maliciousness based on the usage of suspicious permissions. The system uses a multi-level based methodology; we first extract and identify the significant features such as permissions, smali sizes, and permission rates from a manually collected dataset of 10,000 applications. Further, we employ various machine learning models to categorize the Apps into their malicious or benign categories. Through extensive experimentations, the proposed method successfully identifies the 5× most significant features to predict malicious Apps. The proposed method outperformed the existing techniques by achieving high accuracies of malware detection i.e., 89.7% with Support Vector Machine, 89.96% with Random Forest, 86.25% with Rotation Forest, and 89.52% with Naïve Bayes models. Moreover, the proposed method optimized up to ~77% of the feature set as compared to the recent approaches, while improving the evaluation metrics such as precision, sensitivity, accuracy, and F-measure. The experimental results show that the proposed system provides a high level of symmetry between irrelevant permissions and malware Apps. Further, the proposed system is promising and may provide a low-cost alternative for Android malware detection for malicious or repackaged Apps.

Keywords: malware detection; repackaged applications; suspicious permissions; static malware analysis

Citation: Akbar, F.; Hussain, M.; Mumtaz, R.; Riaz, Q.; Wahab, A.W.A.; Jung, K.-H. Permissions-Based Detection of Android Malware Using Machine Learning. *Symmetry* **2022**, *14*, 718. https://doi.org/10.3390/sym14040718

Academic Editor: Zhixun Su

Received: 9 February 2022
Accepted: 22 March 2022
Published: 2 April 2022

1. Introduction

Android is one of the commonly used smartphone operating systems and is currently occupying around 70.97% of the market share (Mobile Operating System Market Share Worldwide https://gs.statcounter.com/os-market-share/mobile/worldwide/, available online: 8 February 2022). According to the recent statistics (App Download and Usage Statistics https://www.businessofapps.com/data/app-statistics/, available online: 8 February 2022), there are around 2.56 million Applications that can be downloaded from the official App stores, while many can be downloaded from the other sources. The statistics show that Android is an absolute leader in the smartphone application market with a daily increase of Apps. The most attractive feature offered by the Android platform is the provision of feature-rich applications for a wide range of users. Moreover, to facilitate users as well as

the developers of the Apps, Google's Play Store has adopted an open-source policy for App availability and provided great tolerance to App verification at the time of release, increasing the popularity of the platform.

Unfortunately, this popularity and the ease of App distribution have spurred the interest of many cyber-criminals across the globe. Reports show that nearly 97% of mobile malware prey on Android devices. In the second quarter of 2021, nearly 1.45 million new Android malware Apps were detected (IT threat evolution Q2 2021. Mobile statistics https://securelist.com/it-threat-evolution-q2-2021-mobile-statistics/103636/, available online: 8 February 2022), which shows that new malware is being produced every few seconds. These malicious Apps are designed for performing different types of offenses in the form of worms, exploits, Trojans, viruses, and many more. Some of these applications are released intentionally in many variants to target a larger audience, making them harder to be detected (Fidelis Threat Intelligence Report—February/March 2021 https://fidelissecurity.com/resource/report/fidelis-threat-intelligence-report-february-march-2021/, available online: 8 February 2022). The intrusion of malicious Apps into the mobile platform has drawn a wide consideration from academia as well as the industry. To address these elevating concerns, analysts and researchers across the world have developed and used various approaches to design efficient Android malware detection tools using different methodologies [1–23].

Generally, the malicious APKs detection mechanisms consist of dynamic and static analysis. The dynamic aspect of the analysis deals with the runtime behavior of applications during their execution phases against selected test cases. However, the static analysis is performed in a non-runtime environment concerning examining the source code, byte code, or application binaries and analyzing the meta and auxiliary information for signs of security vulnerabilities [24,25]. Since the dynamic approach involves detailed analysis of the applications it is considered to be an accurate method for detection, however, it requires high computational cost. Further, the analysis is performed after the execution of the APKs, unlike the static approach. Due to this reason, static analysis is considered to be faster and more helpful in developing an initial view of the APKs based on their expected behaviors.

Static analysis involves an encyclopedic spectrum of strategies and techniques seeking to recognize the runtime demeanors of a software preceding its execution. In a security contexture, the motivation is naturally to segregate conclusively malicious or repackaged Apps ahead of their installs and executions. Static analysis flags an App as malicious on the authority approximation of its possible runtime behaviors. These approximations generally result from the methods encompassing permissions, code analysis or API calls, Intents, App components, file property, native code, etc., [17].

Generally, Android employs the permission-based security model to protect users' information, or to restrict Apps from accessing users' sensitive information. Typically, Android Apps permissions are exploited because they are considered to be one of the most fundamental and paramount security evaluation mechanisms on the Android platform. Therefore, it is almost impossible to take planned action, making the permission scanning a critical step for malware detection without being granted explicit permission. Android Apps request a list of permissions to be granted before providing its functionality to the users. Multiple permissions may reflect some adverse behaviors when used together. For example, the App may procure the users' SMS information and then disseminate it out over the Internet when an App asks for network permission together with the SMS access permission. It means that Android permissions are widely used and considered one of the most effective static features.

In the literature, various malicious APK detection approaches employed the permission and API calls with existing machine learning and deep learning models with their merits and demerits [4,5,7–11,13,17–19,21,22,26]. Recently, a detection technique has also been evaluated on cloud computing [19]. Similarly, a semantic analysis-based approach named SWORD [20] has been proposed that identifies evasion-aware malicious Apps using Asymptotic Equipartition Property (AEP).

In this study, we primarily focus on permissions-based detection. The proposed scheme is inspired by the method of Hui-Juan Zhu et al. [9] that consists of a permission-based malicious APKs detection strategy. The motive of the proposed scheme is to improve the performance of malicious detection APKs while reducing the number of selected permissions for classification purposes. The scheme of Hui-Juan Zhu et al. [9] only employed the Support Vector Machine (SVM) and Rotation Forest classifiers. On the other hand, the proposed Permission-based Malicious Apps detection (PerDRaML) strategy successfully explored and improved the detection accuracies by finding the best classifiers already in practice such as Random Forest and Naive Bayes. The main aim of the proposed study is to identify the minimal and effective permissions set as compared to the existing methods while achieving high classification accuracies.

In the proposed scheme, we first decompile Apps using AndroGuard for the permission extraction process. Further, the permissions are extracted from dataset applications collected from VirusShare (http://virusshare.com/, 28 January 2021) and the Official Play Store. The extracted permissions are then filtered using the feature importance technique and combined with other features—permission rates and the App size metrics. After generating the permission feature set, then various machine learning models for detecting malicious Apps are employed. The contributions of the proposed scheme are as follows:

- A lightweight malware detection system.
- Manual collection of a new ~10,000 malware and benign APKs dataset, publicly available at GitHub [27].
- Optimization of permission feature set (~77%) as compared to existing techniques.
- Improved detection accuracy up to 90% using standard ML techniques—SVM, Random Forest, and Naïve Bayes based classifiers.
- Fully functional source code of the proposed scheme readily available [28] for future research.

The remainder of this paper is organized as follows: Section 2 presents the literature review and Section 3 consists of the proposed methodology. Experiments and performance comparisons are reported in Section 4. Finally, the conclusion and future work is discussed in Section 5.

2. Literature Review

In this section, we will examine the existing malicious App detection schemes more towards the domain of static analysis. Feizollah et al. [1] employed static analysis to apply characteristic-based methodology. They argued the effectiveness of using android intents and extracted intents from different Android applications according to the features for the malware classification. Feizollah et al.'s approach can achieve a 91% detection rate with intents and 95.5% with the combination of intents and permissions. However, it is justified that the intents are not an ultimate solution and should be used in conjunction with other characteristics. Similarly, Nisha et al. [2] proposed the detection of repackaged Android malware using mutual information and chi-square techniques for the selection of features. Random forest classifier was able to achieve the highest accuracy of 91.76% among employed classifiers. However, Nisha et al.'s technique is focused on the 88 uniquely identified permissions for the analysis, which can be further optimized to include only the harmful ones.

Similarly, Sandeep [3] extracted information from the applications and performed Exploratory Data Analysis (EDA). The EDA approach focuses on the detection of malware using deep learning techniques during the installation process. Sandeep's detection framework employed several options like permissions to mirror the behaviors of the applications. It achieved 94.6% accuracy when Random Forest was used as the classifier. The approach uses 331 features for the classification which can further be optimized. Li et al. [4] suggested a permission-based detection system called SIGPID on the permission usage analysis. Li et al. employed a multi-level data pruning technique for the selection of features. Using three levels of pruning—Permission Ranking with Negative Rate (PRNR),

Support Based Permission Ranking (SPR), and Permission Mining with Association Rules (PMAR)—they could identify 22 significant permissions. The SIGPID technique has reached ~90% precision, accuracy, recall, and F-measure by employing SVM classifier.

Similarly, Wang et al. [5] performed a Multilevel Permission Extraction (MPE) approach where they focused on automatically identifying the permission interaction that helps to distinguish between the benign and the malicious applications. Their dataset included 9736 applications from each category set—benign and malicious—and experimental results show that a detection rate of 97.88% was achieved. In Fan et al. [6], an approach called fregraphs was proposed, where constructed frequent subgraphs represent the typical behaviors of malware belonging to the same family. Fan et al. proposed FalDroid, which is a system based on fregraphs for detection. Experimental results showed that FalDroid can classify up to 94.2% of malware samples into their respective categories on average within 4.6 s per App. Similarly, Fatima et al. [7] developed a host-server-based methodology for malware detection. Fatima et al.'s research is focused on the extraction of the application features such as permissions, App components, etc. Further, they were sent to the remote server for analysis, where a machine learning random classifier was applied for the malware classifications. Fatima et al.'s host-server approach can mitigate the computational overhead and resource constraints while gaining over 97% accuracy, but the scheme requires a complete server scale infrastructure to handle the real-time request. Further, the proposed scheme hasn't discussed the security of data involved in the process.

The DroidSieve method was proposed by Guillermo et al. [11]. They have analyzed the syntactical characteristics of the Apps for detection and classification. The DroidSieve scheme collected a listing of API calls from the code. Further, it collected the requested permissions with a set of the existing application components in the context of static features. Guillermo et al. have collected the feature sample for ~100,000 Apps from both classes and then fed it to ML algorithms such as ExtraTrees, SVM [12], and XGBoost [29]. As a result, the DroidSieve scheme achieved 99.44% detection accuracy with zero false positives. According to the authors, the approach is not robust against mimicry attacks and may degrade due to concept drift.

Similarly, Qiao et al. [13] also proposed a similar detection approach based on the API calls and the permissions. Qiao et al. note that the requested permissions within the AndroidManifest.xml file alone may be an over-approximation since some applications usually request excess permissions. To mitigate this, Qiao et al. decompiled the dex byte code into Java source code to extract API calls and map the used permissions in order to generate the feature set. Experimental results from 6260 applications, the SVM [12], Random Forest [30], and Artificial Neural Networks (ANN) [31] showed as detection rates between 78.40% and 94.98%. Their approach requires mapping of permissions with its usage increasing overhead. In another approach, Wu et al. [14] proposed a system named DroidMat which considers multiple static features such as permissions, intents, and API calls with the usage in the components such as service and activity. Wu et al. noticed that a combination of *K*-Means [15] and *k*-nearest neighbors (KNN) [16] with $k = 1$ as the classification provides optimal results. However, DroidMat possesses just one sample of the malware for the various Android malware families and an incontrovertible fact that limits DroidMat's inferring ability of a malware's behavior. Similarly, Sun et al. [8] used static analysis in the study and proposed a collection of program characteristics consisting of sensitive API calls and permissions, and performed evaluations using the Extreme Learning Machine (ELM) technique. Sun et al.'s aim consists of detection which involves minimal human intervention. Sun et al. implemented an automated tool called WaffleDetector. The proposed tool showed ~97.14% accuracy using the ELM technique. However, the results are based on a small dataset of only 1049 applications from third-party or unofficial stores.

Zhu et al. [9] proposed a low-cost system for malware detection by extracting a set of system events, permissions, and sensitive APIs and calculating the permission rate according to the key features. Zhu et al.'s approach employed the ensemble Rotation Forest (RF) to construct a model for the classification of malicious and benign APKs. The approach

yielded over 88% detection accuracy which is almost 3.3% higher when compared with SVM classifier. However, Zhu et al.'s technique is evaluated on the limited APKs dataset. Further, the proposed feature set can be optimized and evaluated with other ML classifiers to improve detection accuracy. For a quick review, Table 1 is designed to summarize the existing approaches.

Table 1. A brief summary of existing static analysis based malicious APK detection approaches.

Reference	Features	No. of Features	No. of Samples	Classifier/Algorithm	Accuracy	Suggestions
Feizollah et al. [1], 2017	Intents (explicit and implicit), Permissions	Variable	7406 (B:1846, M:5560)	Bayesian Network using k-fold cross-validation	95.5%	To improve the accuracy, the intents can be utilized in conjunction with other characteristics
Nisha et al. [2], 2018	Permissions	88	-	KNN, SVM, DT, Naïve Bayes and Random Forest	92.94%	Permissions can be reduced by exploring the most relevant ones (i.e., Li et al. [4], 2017).
Sandeep [3], 2019	Permissions	331	-	Random Forest	94.65%	Other classifiers can be explored to improve the detection accuracies. Further, the features can be optimized while incorporating the resistance against mimicry attacks, App cloning, or adware
Li et al. [4], 2017	Ranking, support-based, and multi-level Permissions Pruning	22	-	SVM	>90%	Unknown samples were collected. The scheme should be tested on larger datasets.
Wang et al. [5], 2019	Permission Interactions	25	9736	MN, SVM, D-Tree, and Random Forest	97.9%	-
Fan et al. [6], 2018	Fregraphs to represent common behaviors of malware	Variable	Multiple datasets	-	94.2%	-
Fatima et al. [7], 2020	Frequent sub-graphs	674	50,000 (B:25,000, M:25,000)	Random classifier by server client arch.	>97%	The security of host vs. server communications should be evaluated.
Sun et al. [8], 2017	Permission, API calls	83	1049 (B:524, M:525)	Extreme Learning Machine	97.14%	Dataset only limited to Chinese markets
Zhu et al. [9], 2018	Permission, API calls	22	2130 (B:1065, M:1065)	SVM and Rotation Forest	>88%	Evaluation on limited APKs dataset and as well as limited classifiers was explored
Guillermo et al. [11], 2017	Permission, API calls	-	100,000	ExtraTrees, SVM, RF, XGBoost	99.82%	-
Qiao et al. [13], 2016	Permission, API calls	-	6260	SVM, Random Forest, ANN	78.40–94.98%	-
Wu et al. [14], 2012	Permission, API calls	-	1738	K-Means, k-nearest neighbors	97.87%	Evaluation on limited APKs dataset

In the aforementioned research, most of the techniques employed the permission-based feature set for the classification of benign and malicious APKs. Permissions based techniques are generally employed for the detection of maliciousness as they provide fast and almost high detection accuracy. Since the analysis is performed before the App installs, any harm to the device is prevented. The permissions can perform a key role in the faster detection of the malware. However, there is a lack of employment of minimal effective permissions to improve detection accuracy. Further, minimizing ineffective permission features may reduce the computation complexity.

Therefore, we propose a scheme that can take the power of the permissions and use it to employ a computation effective and faster detection approach. In the next section, we will discuss the proposed scheme to overcome the limitation of existing methods.

3. The Proposed Methodology

In this section, we will discuss the proposed scheme that is inspired by and an improvement of Zhu et al.'s [9] technique. The proposed scheme enhanced the detection accuracy while reducing the number of permissions based on the effectiveness of the permission's

features. The proposed Permission-based Malicious Apps detection (PerDRaML) system extracts permission usage from the application packages, but rather than analyzing all requested permissions, PerDRaML mainly targets the selected set of permissions that are effective in distinguishing and improving the rate of malware detection. For the classification, the proposed scheme employed the Support Vector Machine (SVM), Rotation Forest, Naïve Bayes, and lastly the Random Forest classifiers. We have selected the permissions based on notable influence on the malware detection potency. The proposed research consists of the following major components:

I. Collecting Malicious and Benign APKs
II. Constructing/Identifying the Features Set
III. Filtering, Finalizing, and Extracting the Permissions (Features) Dataset
IV. Employing the Supervised ML algorithms to Classify the Android Malware

The proposed PerDRaML strategy is shown in Figure 1a,b. Figure 1a depicts the collections, decompiling, constructing, and filtering of APKs parameter features for a dataset. Figure 1b depicts the evaluation of the framework and how APKs will be evaluated on classifiers to distinguish by benign and malicious. The key components are as follows.

3.1. Malicious and Benign Samples Collection

For the dataset, we have collected Android applications from two sets of android families, malicious and benign as a general. We have gathered (~5000) malicious Apps from VirusShare (http://virusshare.com/, 28 January 2021), a well-known Android malware database. VirusShare collects applications from different malware families from time to time and makes them available as zip archives. These files can be downloaded using any torrent client. The benign Apps (~5000) have been collected from the official App store (Google's Play Store) using our Python-based script available at [32]. To provide more diversity to the dataset, benign APKs have been collected from different Play Store's application categories. The total number of APKs consists of 10,000 samples, half of which (5000) belong to each category [28]. These samples are used in the research to carry out cross-validation experimentations as well as for training data in order to evaluate the effectiveness of the proposed methodology.

3.2. Constructing the Parameter Feature Set

The first step of building and classifying the classifier models is the collection of relevant key permissions from a dataset. Generally, the features such as permissions requested by an application are enlisted in the Android application package (APK's) *AndroidManifest.xml*. For permission extraction, we have adopted Androguard [23] to decompile the 10,000 applications for our dataset. We extracted various types of possible permissions to construct the feature set list i.e., *permissions, permission rate*, and the *smali sizes* of the applications to carry out static analysis and to explicitly grasp the behavior of each App.

3.3. Filtering, Finalizing, and Extracting the Core Features

After identifying the list of permissions, this section aims to determine the most significant permissions that can be used for distinguishing Apps from malicious and benign ones. To select the significant key permissions for malware detection, we have explored and evaluated Google's dangerous permission list [2] and the Zhu et al. [9] permission set as shown in Table 2. From Table 2, it is observed that Zhu et al. [9] overlapped the nine Google permission features for its evaluation while employing an additional feature named *permission rate*. The selection of effective permissions is stated in the next section.

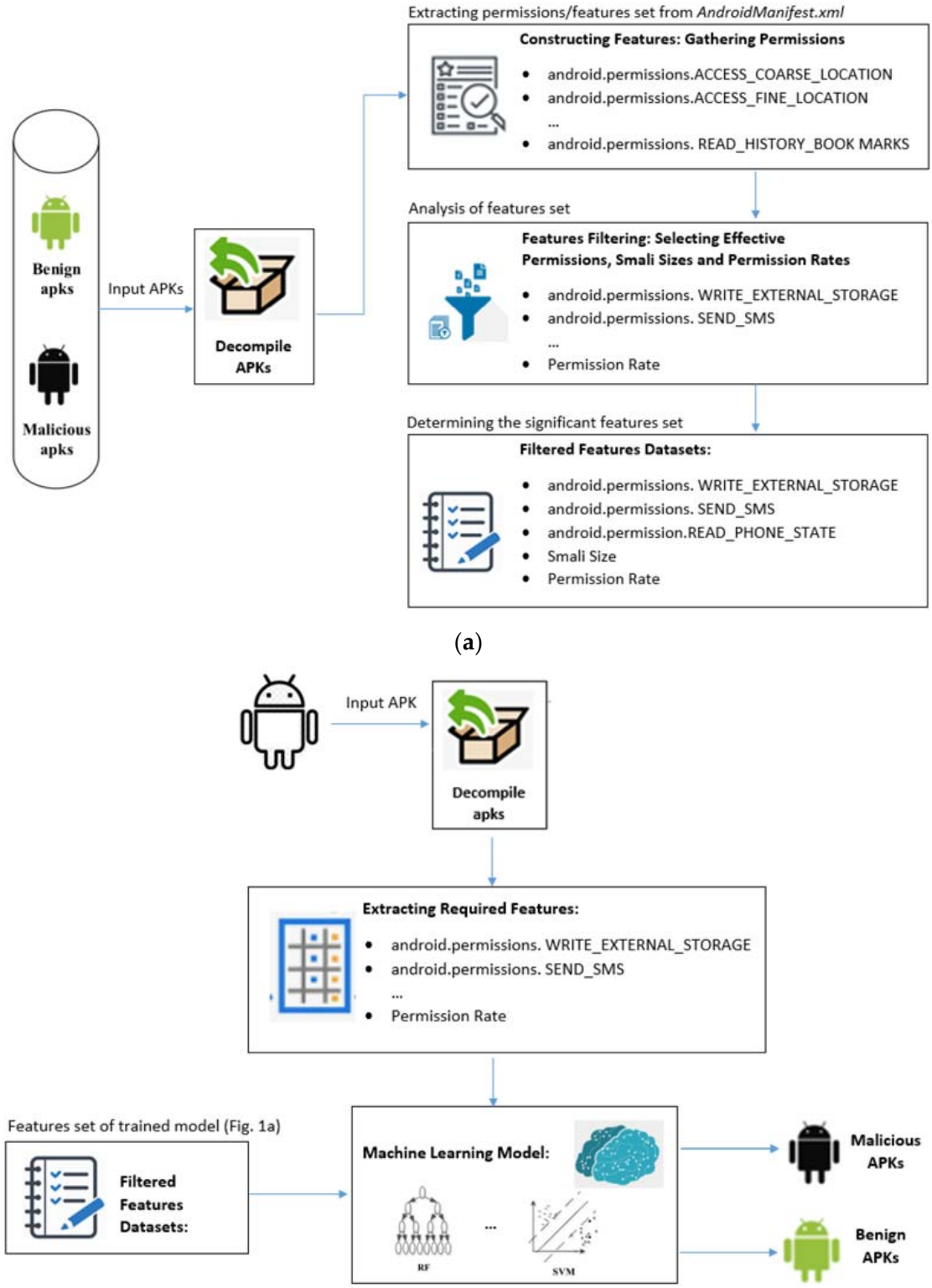

Figure 1. (**a**) Flow diagram of creating features, constructing datasets for malicious and benign APKs. (**b**) Flow diagram of malicious and benign APKs classifications.

Table 2. Feature importance of dangerous permissions identified by Google and Zhu et al. [9] with additional metrics.

S. No	Features	Source	Importance%	S. No	Features	Source	Importance %
1	READ_PHONE_STATE	G	0.31892	15	ACCESS_NOTIFICATION_POLICY	G	0.00337
2	Smali Size	R	0.22674	16	WRITE_CONTACTS	G	0.00265
3	Permission rate	R	0.20376	17	READ_CALENDAR	G	0.00235
4	WRITE_EXTERNAL_STORAGE	G + R	0.06948	18	READ_CALL_LOG	G	0.00200
5	ACCESS_COARSE_LOCATION	G	0.05323	19	WRITE_CALENDAR	G	0.00175
6	ACCESS_FINE_LOCATION	G	0.01932	20	INSTALL_PACKAGES	G + R	0.00151
7	RECORD_AUDIO	G	0.01923	21	SET_ALARM	G + R	0.00090
8	READ_EXTERNAL_STORAGE	G	0.01588	22	BODY_SENSORS	G	0.00080
9	SEND_SMS	G + R	0.01268	23	WRITE_SECURE_SETTINGS	G + R	0.00077
10	CAMERA	G	0.01024	24	WRITE_CALL_LOG	G	0.00069
11	RECEIVE_SMS	G + R	0.01009	25	UPDATE_DEVICE_STATS	G + R	0.00016
12	GET_ACCOUNTS	G	0.00953	26	READ_HISTORY_BOOKMARKS	G + R	0.00002
13	READ_SMS	G	0.00743	27	WRITE_HISTORY_BOOKMARKS	G + R	0.00000
14	READ_CONTACTS	G	0.00650	-	-	-	-

Google: G; Benchmark, Zhu et al. [9]: R.

3.3.1. Selection of Effective Permissions

To select the minimum set of permissions we filter out the permissions that are less impactful for the detection. For this, we employed different combinations of Google's dangerous permission list (from Table 2) while also incorporating the feature importance property. Feature importance is a measure that helps generate simpler and faster prediction models using fewer inputs [33,34]. We use Random Forest-based feature importance to enlist the important permissions (Table 2). We have defined a threshold measure of 0.1 to select the most impactful ones by ignoring the permissions that show importance lower than 0.1. Through the various combinations of feature set experiments, we finally identify the most significant permission list as shown in Table 3.

Table 3. The proposed identified parameter features.

Type	No	Name
Permissions	1	android.permission.READ_PHONE_STATE
	2	android.permission.WRITE_EXTERNAL_STORAGE
	3	android.permission. ACCESS_COARSE_LOCATION
Metrics	1	Smali Size
	2	Permission Rate

3.3.2. Generating Dataset

The permission information is translated into a dataset in the binary format where '1' specifies an App requesting the permission, and '0' indicates the opposite. The permission lists extracted from malicious and benign Apps, represented in the binary format, are combined to make one holistic dataset for analysis.

However, the direct comparison of Zhu et al. [9] with proposed selected features can be seen in Table 4. In order to better classify the malware, we also accumulate some other metrics for benign and malware training datasets such as *permission rate* and *smali size*. In this paper, *permission rate* (*pr*) is adopted from [9] as one of the detection metrics defined by the formula in Equation (1):

$$pr = \frac{pn}{ss} \quad (1)$$

where *pn* is the total amount of permissions requested by the App and *ss* represents the size of the application's *smali* files obtained after decompiling the APK. The *smali size* (*ss*) metric is based on the assumption that the malware Apps usually request a higher set of permissions and also require more implementation to abuse the privileges offered by them, thus increasing the size of the App. Due to this reason, this metric can be used as an identifier for the repackaged applications.

Table 4. Comparison of the proposed and Zhu et al. [9] methods' features.

	Permissions/Features	Zhu et al. [9]	Proposed Method
Permission	WRITE_EXTERNAL_STORAGE	✓	✓
	UPDATE_DEVICE_STATS	✓	X
	SET_ALARM	✓	X
	INSTALL_PACKAGES	✓	X
	WRITE_HISTORY_BOOKMARKS	✓	X
	WRITE_SECURE_SETTINGS	✓	X
	READ_HISTORY_BOOKMARKS	✓	X
	RECEIVE_SMS	✓	X
	SEND_SMS	✓	X
	READ_PHONE_STATE	X	✓
	ACCESS_COARSE_LOCATION	X	✓
APIs & URLs	sendTextMessage()	✓	X
	getMessageBody()	✓	X
	getSubscriberId()	✓	X
	getLine1Number()	✓	X
	getLastKnownLocation()	✓	X
	content://com.android.contacts	✓	X
System Event	DATA_SMS_RECEIVED	✓	X
	BATTERY_CHANGED	✓	X
	AIRPLANE_MODE	✓	X
	SMS_RECEIVED	✓	X
	c2dm.intent.RECEIVE	✓	X
	QUICKBOOT_POWERON	✓	X
Metrics	Permission Rate	✓	✓
	Smali Size	X	✓
Percentage (Feature Usage)		88.0%	20.0%

3.4. Malware Detection by Employing Machine Learning Classification Models

In this section, we employed supervised machine learning classifiers that correctly predict malicious Apps with minimum false positives. The overall strategy of the proposed approach is carved up into two parts; the first one consists of Permissions Extraction (with APK decompiling, matrix calculations, and finalizing the dataset features), and the second one consists of training and validation of the supervised learners using the aforementioned datasets with various ML algorithms.

The proposed dataset consists of 10,000 samples made up of 5000 samples from each category. In experiments, we applied a similar training and testing strategy technique as Zhu et al. [9] to create uniformity. For this, we employed the 10-fold crossed valida-

tion technique for the classification of each model to avoid the over-fitting problem in the dataset.

4. Performance Evaluation

In this section, we will evaluate the effectiveness of the proposed PerDRaML malware detection approach. We have selected conventional machine learning algorithms; SVM, Random Forest, and Naive Bayes models with the benchmark (Rotation Forest) to evaluate the proposed scheme. Initially, the Rotation Forest classification model is evaluated on both the proposed and Zhu et al. [9] feature set model. Further, we compared the detection accuracy results on three different models i.e., SVM, Random Forest, and Naive Bayes with the proposed and Zhu et al. [9] approaches.

The proposed PerDRaML approach only employed the most 5× significant permissions instead of the 22× different feature sets that were used in the benchmark approach [9]. Table 5 shows the results of the benchmark approach where the detection accuracies are around 84.93 ± 1.8%, 88.26 ±1.73% on SVM and Rotation Forest classifiers. However, the proposed method outperformed the benchmark approach [9] by achieving classification accuracies of approximately 89.70% with minimum permissions set. In the next section, we will discuss the evaluation measures in detail.

Table 5. The performance results of Zhu et al. approach [9] on SVM, rotation forest with 10-fold cross-validation.

Classifier	Test Set	Precision (%)	Sensitivity (TPR, %)	Accuracy (%)	AUC
SVM	1	85.71	87.27	85.91	0.85
	2	87.27	83.48	84.51	0.87
	3	81.48	83.02	82.16	0.83
	4	77.06	85.71	81.69	0.83
	5	87.62	85.98	86.85	0.88
	6	81.82	88.24	84.98	0.86
	7	82.30	89.42	85.45	0.89
	8	84.40	87.62	85.92	0.88
	9	88.39	83.19	84.51	0.85
	10	85.29	87.88	87.32	0.86
Average (%)		84.13 ± 3.5%	86.18 ± 2.3	84.93 ± 1.8	0.86 ± 0.02
Rotation Forest	1	88.18	88.18	87.79	0.88
	2	88.03	89.57	87.79	0.90
	3	86.11	87.74	86.85	0.86
	4	84.62	89.80	87.79	0.90
	5	88.35	85.05	86.85	0.88
	6	89.11	88.24	89.20	0.89
	7	89.81	93.30	91.55	0.93
	8	87.00	82.86	85.45	0.86
	9	91.30	88.24	88.73	0.89
	10	89.11	90.91	90.61	0.90
Average (%)		88.16 ± 1.80	88.4 ± 2.76	88.26 ± 1.73	0.89 ± 0.02

4.1. Evaluation Measures

For evaluation, we followed the standard evaluation metrics: Sensitivity, Precision, Accuracy, Area Under Curve (AUC), and the Receiver Operating Characteristic (ROC). Correspondingly, the following formulas represent their definitions.

$$Accuracy = \frac{T\,P + T\,N}{T\,P + F\,P + T\,N + F\,N}$$

$$Sensitivity = \frac{T\,P}{T\,P + F\,N}$$

$$Precision = \frac{T\,P}{T\,P + F\,P}$$

$$F1 - Score = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

where true positive (*TP*) represents the number of positive testing samples that are accurately predicted as positive, false positive (*FP*) is the number of negative testing samples that are falsely predicted as positive, true negative (*TN*) is the number of negative testing samples accurately predicted as negative, and false negative (*FN*) is the number of positive testing samples falsely predicted as negative.

4.2. Evaluating PerDRaML Effectiveness

To evaluate the effectiveness of the proposed scheme, we employed various classification models to illustrate the generality of PerDRaML. In experiments, we employed the cross-fold validation technique to the classifier models from the prospect of stability and in order to elude the over-fitting. We compared our experiments using the 10-cross validation. Particularly, the samples are randomly divided into ten disjoint copies, each classifier taking one of the copies as the test set and the remaining nine as the training set to construct the model. The results of the 10-folded classification can be seen in Tables 6–9 with respective ROC figures from 2 to 5.

Table 6. The proposed scheme results on SVM classifier with 10-fold cross-validation.

Classifier	Test Set	Precision (%)	Sensitivity (TPR, %)	F1—Score (%)	Accuracy (%)	AUC
SVM	1	89.63	89.65	89.60	89.60	0.95
	2	89.66	89.64	89.60	89.60	0.95
	3	90.01	89.98	89.99	90.00	0.95
	4	89.07	89.01	89.00	89.00	0.93
	5	90.63	90.63	90.60	90.60	0.95
	6	90.46	90.21	90.27	90.30	0.95
	7	89.11	88.82	88.87	88.90	0.94
	8	90.63	90.59	90.50	90.50	0.95
	9	88.81	88.81	88.80	88.80	0.94
	10	89.73	89.66	89.68	89.70	0.94
Average (%)		89.77	89.70	89.69	89.70	0.94

Table 7. The proposed scheme results on rotation forest classifier with 10-fold cross-validation.

Classifier	Test Set	Precision (%)	Sensitivity (TPR, %)	F1—Score (%)	Accuracy (%)	AUC
Rotation Forest	1	85.49	85.49	85.49	85.50	0.86
	2	86.50	86.39	86.39	86.40	0.88
	3	86.38	86.43	86.39	86.40	0.87
	4	85.31	85.33	85.30	85.30	0.87
	5	87.62	87.62	87.60	87.60	0.89
	6	86.59	86.58	86.58	86.60	0.88
	7	86.20	86.19	86.20	86.20	0.88
	8	87.65	87.67	87.66	87.70	0.89
	9	85.81	85.81	85.80	85.80	0.87
	10	84.94	84.90	84.92	85.00	0.87
Average (%)		86.25	86.24	86.23	86.25	0.88

Table 8. The proposed scheme results on random forest classifier with 10-fold cross-validation.

Classifier	Test Set	Precision (%)	Sensitivity (TPR, %)	F1—Score (%)	Accuracy (%)	AUC
Random Forest	1	91.30	91.29	91.30	91.30	0.95
	2	88.12	88.14	88.10	88.10	0.94
	3	89.44	89.28	89.34	89.40	0.94
	4	89.47	89.50	89.48	89.50	0.95
	5	91.51	91.48	91.49	91.50	0.96
	6	89.73	89.70	89.70	89.70	0.95
	7	90.56	90.63	90.58	90.60	0.95
	8	89.09	89.13	89.10	89.10	0.95
	9	89.71	89.63	89.60	89.60	0.95
	10	90.80	90.82	90.80	90.80	0.95
Average (%)		89.97	89.96	89.95	89.96	0.95

From Table 6, the proposed strategy using SVM achieved the average accuracy as 89.7% with 89.77% precision and 89.7% sensitivity. The ROC can also be seen in Figure 2 for SVM based classifier. Similarly, the accuracy using the Rotation Forest classifier on the proposed feature set achieved up to 86.25% with 86.25% precision and 86.24% sensitivity as shown in Table 7. Figure 3 also depicts the ROC of the rotation forest classifier. Similarly, the proposed strategy using Random Forest achieved an average accuracy of 89.96% with 89.97% precision and 89.96% sensitivity as shown in Table 8. The ROC can also be seen in Figure 4 for Random Forest-based classifier. Finally, the Naive Bayes classifier accuracy can be seen in Table 8, where the average detection rate is 89.52% with 89.53% precision and 89.52% sensitivity. The ROC graph of Naïve Bayes is shown in Figure 5.

Table 9. The proposed scheme results on Native Bayes classifier with 10-fold cross-validation.

Classifier	Test Set	Precision (%)	Sensitivity (TPR, %)	F1—Score (%)	Accuracy (%)	AUC
Naive Bayes	1	88.80	88.80	88.80	88.80	0.92
	2	89.25	89.39	89.28	89.30	0.93
	3	89.52	89.49	89.50	89.50	0.92
	4	89.79	89.82	89.80	89.80	0.94
	5	89.25	89.12	89.17	89.20	0.92
	6	90.12	90.10	90.10	90.10	0.93
	7	88.53	88.65	88.49	88.50	0.92
	8	90.60	90.60	90.60	90.60	0.94
	9	89.63	89.49	89.55	89.60	0.93
	10	89.82	89.75	89.78	89.80	0.92
Average (%)		89.53	89.52	89.50	89.52	0.93

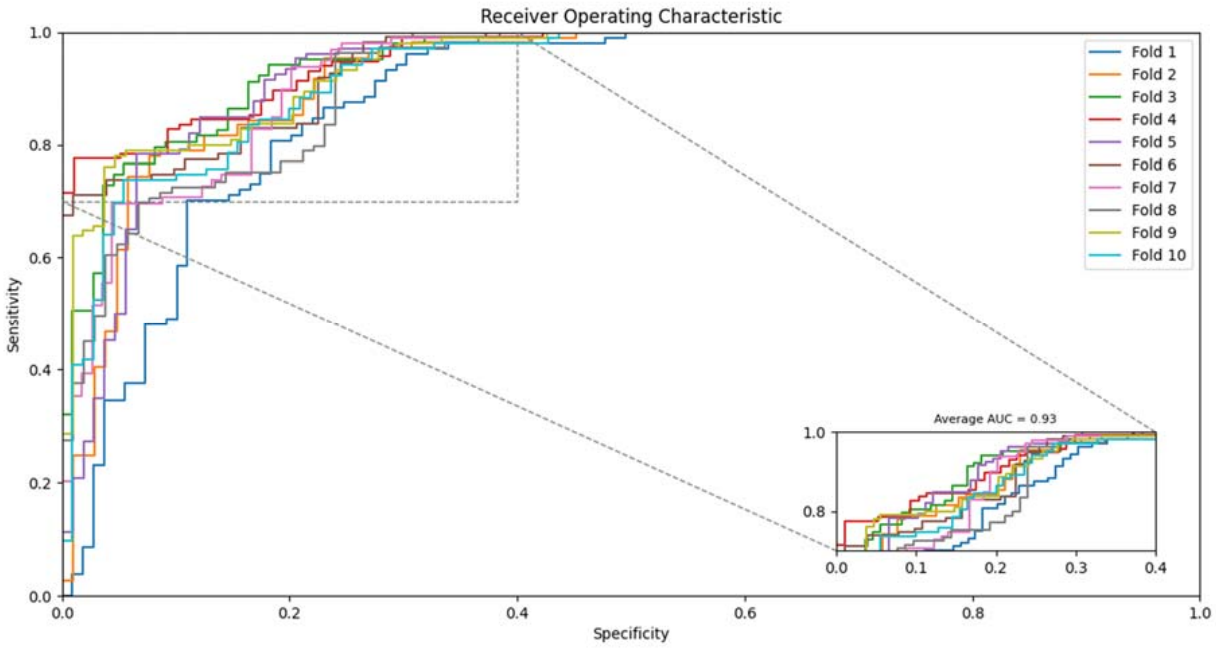

Figure 2. Receiver operating characteristic (ROC) achieved by the proposed method on SVM classifier.

The overall performance comparison of the proposed and Zhu et al. [9] can be seen in Table 10. A graphical representation of the aforementioned evaluation is depicted in Figure 6, where the Random Forest algorithm is considered as the best classifier in terms of detection accuracies. It is also notable that the proposed scheme achieves a similar set of accuracies on SVM and Rotation Forest models as compared to Zhu et al. [9] while incorporating the lower number of permission feature sets resulted in Table 10.

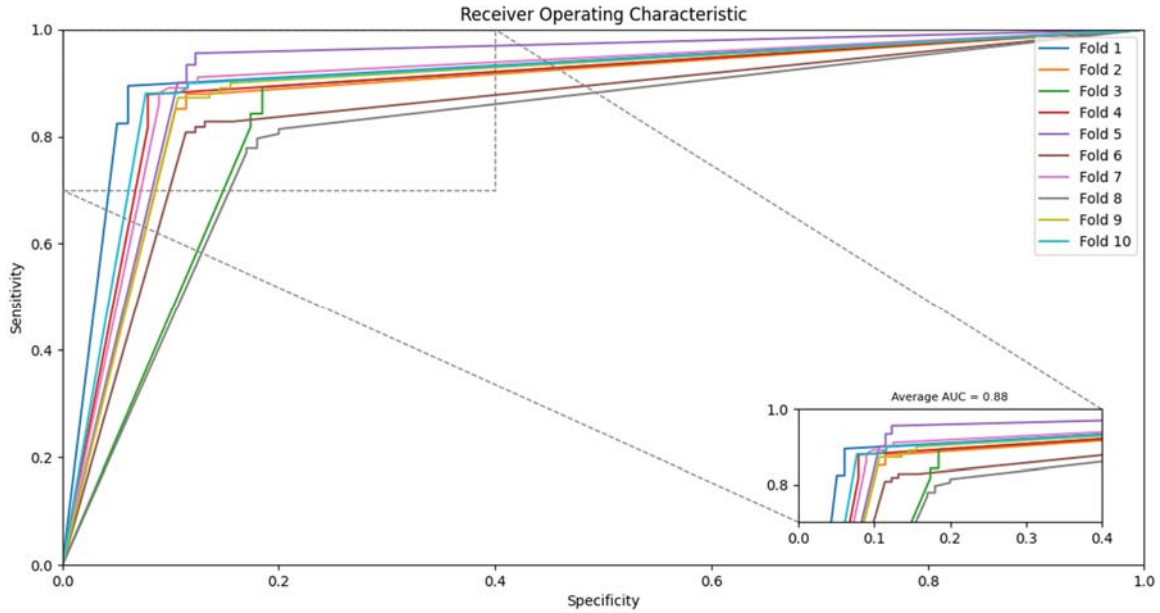

Figure 3. Receiver operating characteristic (ROC) achieved by the proposed method on rotation forest classifier.

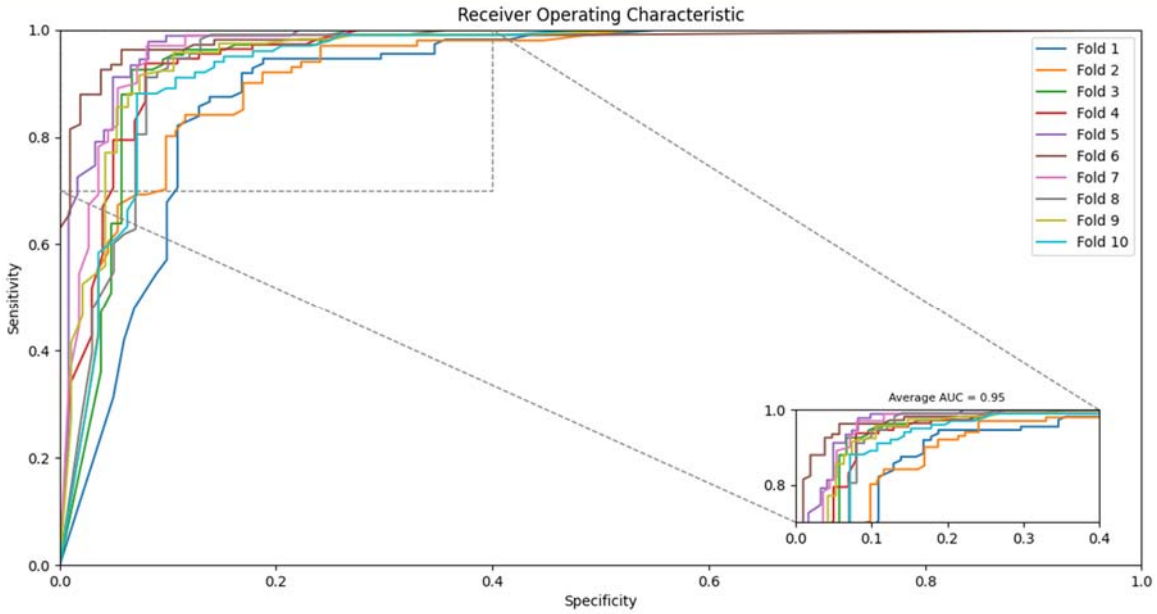

Figure 4. Receiver operating characteristic (ROC) achieved by the proposed method on Random Forest classifier.

Table 10. Performance comparison of the proposed and Zhu et al. [9] using 10-fold cross-validation.

Classifier	Zhu et al. [9] Approach				Proposed Approach			
	# of Features	Precision (%)	Sensitivity (%)	Accuracy (%)	# of Features	Precision (%)	Sensitivity (%)	Accuracy (%)
SVM	22	84.13 ± 3.5	86.18 ± 2.3	84.93 ± 1.8	5	89.7 ± 0.8	89.7 ± 0.85	89.7 ± 0.9
Rotation Forest		88.16 ± 1.80	88.4 ± 2.76	88.26 ± 1.73		86.25 ± 1.3	86.24 ± 1.4	86.25 ± 1.25
Random Forest		-	-	-		89.97 ± 1.25	89.96 ± 1.5	89.96 ± 1.5
Naïve Bayes		-	-	-		89.53 ± 0.75	89.52 ± 1.1	89.52 ± 1.0
Reduction	0%				77.23%			

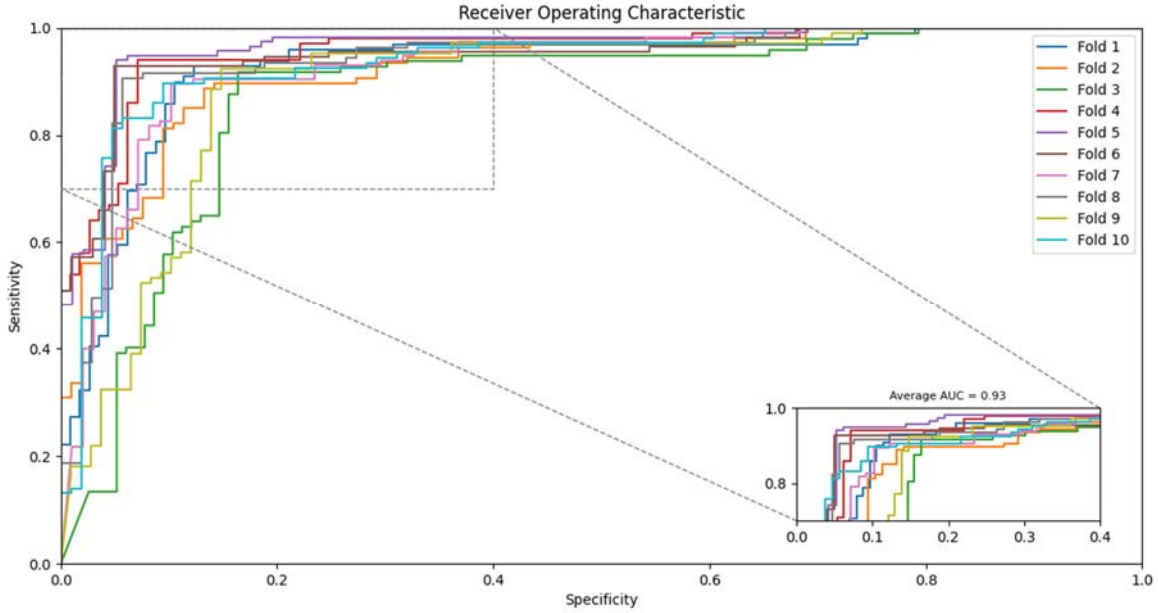

Figure 5. Receiver operating characteristic (ROC) achieved by the proposed method on Naive Bayes classifier.

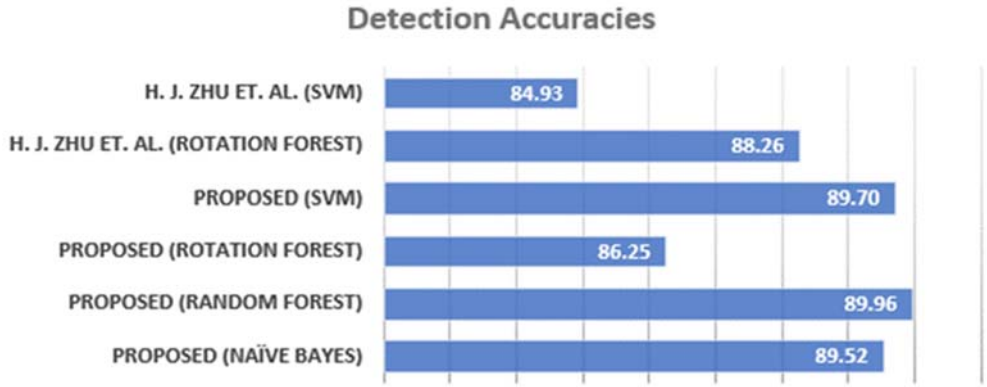

Figure 6. Detection accuracy of the proposed and Zhu et al. [9] schemes on various machine learning classifiers.

It is evident that SVM, Random Forest, and Naïve Bayes classifier using the proposed set of features gives the best accuracy ratings of 89.7 $\pm$ 0.25%. From Table 10, it is also notable that the proposed scheme successfully reduces the number of permissions to 77.2% while maintaining similar detection accuracies. From the above comparison, it can be seen that the proposed approach outperforms the Zhu et al. [9] approach in terms of accuracy, where the Random Forest classifier turned out to be the best performing model.

Meanwhile, we also evaluated the Google-based dangerous permissions feature set using SVM, rotation forest, random forest, and Naïve Bayes classifiers as seen in Table 11. It is also evident that the proposed parameters set has similar detection accuracies while having the minimum 5$\times$ parameters permissions set instead of 22$\times$, around 77.2% fewer features than in Table 10. Similarly, the overall performance comparison of the proposed and other existing approaches in the number of permissions and detection accuracies is shown in Figure 7. It is also depicted that the proposed strategy achieved ~90% detection accuracy while only employing the 5$\times$ permissions as compared to the existing approaches.

Table 11. Detection results using Google dangerous permission on SVM, random forest, rotation forest, and Naïve Bayes classifier using 10-fold cross-validation.

Classifier	Precision (%)	Sensitivity (TPR, %)	F1—Score (%)	Accuracy (%)
SVM	89.79	89.72	89.71	89.72
Rotation Forest	88.10	88.10	88.03	88.04
Random Forest	90.93	90.94	90.92	90.93
Naïve Bayes	76.58	68.59	65.99	68.58

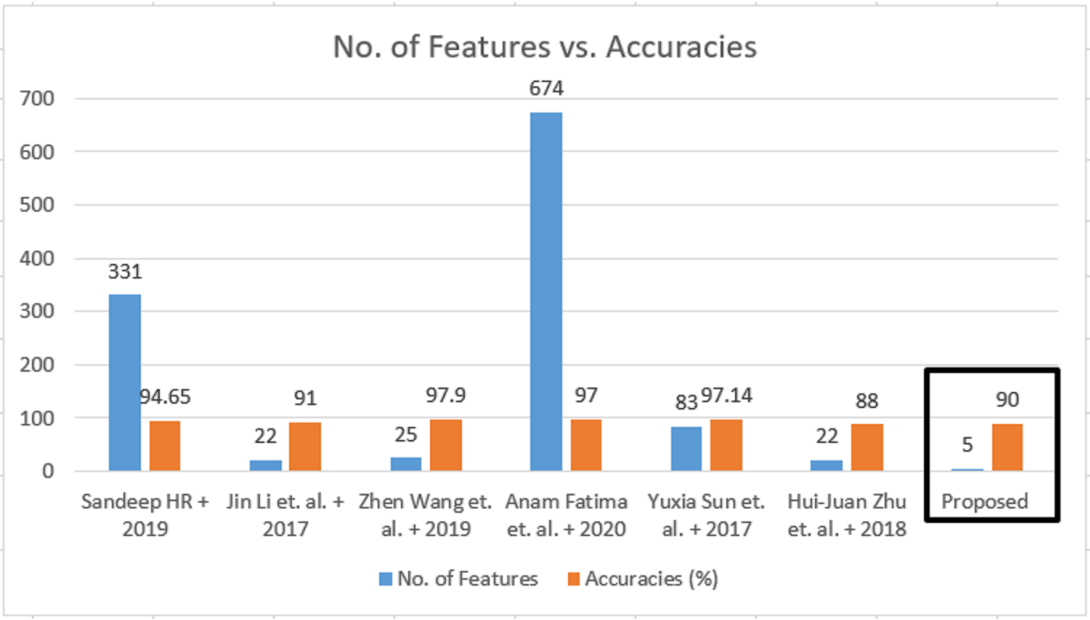

Figure 7. Comparisons of number of permissions and accuracy of the proposed and existing [3–5,7–9] approaches

4.3. Discussion

Malware detection systems are required to have a heightened performance metric which can be achieved by either improving the dataset collection process or by improving the selected features. For high performance, a significant aspect of the proposed method is the selection of significant permissions since they are proven to be an impacting figure considered by the literature. An App needs to acquire user consent to perform the necessary activities, and therefore, making permissions is the main area of focus in our study. Another significance of the proposed method which targets the other aspect of evaluation performance lies in the careful selection of the dataset samples. Moreover, the proposed scheme is trained and validated on a large dataset comprising of 10,000 APKs acquired from authentic Google Play Store and VirusShare sources. Upon comparison of our findings with the existing and benchmark (Zhu et al. [9]) methods, it is observed that the proposed strategy achieves almost similar detection accuracies on SVM and Rotation Forest classifiers by employing the proposed effective permissions set as shown in Table 3. Identifying the lower number of feature sets helps the classifier to achieve the high detection accuracies that can reduce the computation overhead and provide a low-cost malware detection solution.

4.4. Applicability of PerDRaML

The proposed approach can differentiate malicious applications from benign ones that are based on the requested permissions during the installation process. The proposed approach can be utilized in many ways. For example, the proposed scheme can be inserted

as a module to the end devices as an anti-malware application which upon every new installation, extracts the permissions from the APK and passes the data to the trained classifier. The data can be used for classification that is based on the results, either allowing the installation or reporting the detection to the end-users. It can also be implemented on the application stores to categorize the applications being uploaded before becoming available to the general public. The proposed approach can also be utilized by implementing both host-based and market-based implementation to provide further security enhancements

5. Conclusions

Mobile malware has recently been posing a great security threat to the mobile ecosystem. Machine learning algorithms required heightened accuracies to address the security issues that are generally focused on an impactful selection of features as well as the efficient performance of the selected classifiers. In this paper, we have statically analyzed the Android ecosystem to show that it is possible to achieve a higher level of accuracy by reducing the number of permissions while maintaining high efficiency and effectiveness. We have selected a base approach proposed by Zhu et al. where we have trained the Rotation Forest classifier to perform detection on the features selected from the permissions, system API calls and events, and other categories. We extensively studied Zhu et al. and Google's dangerous permission sets through various experiments to identify the most significant number of permission sets to improve the efficiency without compromising the detection accuracy. With the permissions dataset, we evaluated the different classifiers—Random Forest, SVM, Naïve Bayes, and Rotation Forest. The experimentation results showed that all classifiers achieved above 89% accuracy by employing the proposed permission dataset. From the proposed evaluation, we concluded that an effective level of detection accuracy could also be achieved using fewer significant permission sets. In the future, the collected dataset will be enhanced, and the proposed scheme will be evaluated on other supervised and unsupervised (without labeling) machine learning classifiers to improve the detection accuracies.

Author Contributions: Conceptualization and implementation, F.A. and M.H.; validation, Q.R.; investigation, R.M.; writing—draft preparation, A.W.A.W.; writing—review and editing, K.-H.J.; supervision, M.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2021R1I1A3049788) and Brain Pool program funded by the Ministry of Science and ICT through the National Research Foundation of Korea (2019H1D3A1A01101687, 2021H1D3A2A01099390); this work was supported by the National University of Sciences and Technology, Islamabad, Pakistan.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The collected malware dataset and the proposed PerDRaML scheme implementation are available [27,28]. Google Play Python API can be seen in [32].

Acknowledgments: We thank the anonymous reviewers for their valuable suggestions that improved the clarity of this article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Feizollah, A.; Anuar, N.B.; Salleh, R.; Suarez-Tangil, G.; Furnell, S. AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection. *Comput. Secur.* **2017**, *65*, 121–134. [CrossRef]
2. Jannath, N.O.S.; Bhanu, S.M.S. Detection of repackaged Android applications based on Apps Permissions. In Proceedings of the 2018 4th International Conference on Recent Advances in Information Technology (RAIT), Dhanbad, India, 15–17 March 2018; pp. 1–8.
3. Sandeep, H.R. Static analysis of android malware detection using deep learning. In Proceedings of the 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Secunderabad, India, 15–17 May 2019; pp. 841–845.

4. Li, J.; Sun, L.; Yan, Q.; Li, Z.; Srisa-An, W.; Ye, H. Significant Permission Identification for Machine-Learning-Based Android Malware Detection. *IEEE Trans. Ind. Inform.* **2018**, *14*, 3216–3225. [CrossRef]
5. Wang, Z.; Li, K.; Hu, Y.; Fukuda, A.; Kong, W. Multilevel permission extraction in android applications for malware detection. In Proceedings of the 2019 International Conference on Computer, Information and Telecommunication Systems (CITS), Beijing, China, 28–31 August 2019; pp. 1–5.
6. Fan, M.; Liu, J.; Luo, X.; Chen, K.; Tian, Z.; Zheng, Q.; Liu, T. Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis. *IEEE Trans. Inf. Forensics Secur.* **2018**, *13*, 1890–1905. [CrossRef]
7. Fatima, A.; Kumar, S.; Dutta, M.K. Host-Server-Based Malware Detection System for Android Platforms Using Machine Learning. In *Advances in Computational Intelligence and Communication Technology*; Springer: Singapore, 2021; pp. 195–205.
8. Sun, Y.; Xie, Y.; Qiu, Z.; Pan, Y.; Weng, J.; Guo, S. Detecting android malware based on extreme learning machine. In Proceedings of the 2017 IEEE 15th Intl Conf on Dependable, Autonomic and Secure Computing, 15th Intl Conf on Pervasive Intelligence and Computing, 3rd Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), Orlando, FL, USA, 6–10 November 2017; pp. 47–53.
9. Zhu, H.-J.; You, Z.-H.; Zhu, Z.-X.; Shi, W.-L.; Chen, X.; Cheng, L. DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing* **2018**, *272*, 638–646. [CrossRef]
10. Faruki, P.; Laxmi, V.; Bharmal, A.; Gaur, M.; Ganmoor, V. AndroSimilar: Robust signature for detecting variants of Android malware. *J. Inf. Secur. Appl.* **2015**, *22*, 66–80. [CrossRef]
11. Suarez-Tangil, G.; Dash, S.K.; Ahmadi, M.; Kinder, J.; Giacinto, G.; Cavallaro, L. Droidsieve: Fast and accurate classification of obfuscated android malware. In Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, AZ, USA, 22–24 March 2017; pp. 309–320.
12. Tong, S.; Chang, E. Support vector machine active learning for image retrieval. In Proceedings of the ninth ACM international conference on Multimedia, New York, NY, USA, 30 September–5 October 2011; pp. 107–118.
13. Qiao, M.; Sung, A.H.; Liu, Q. Merging permission and api features for android malware detection. In Proceedings of the 2016 5th IIAI international congress on advanced applied informatics (IIAI-AAI), Kumamoto, Japan, 10–14 July 2016; pp. 566–571.
14. Wu, D.-J.; Mao, C.-H.; Wei, T.-E.; Lee, H.-M.; Wu, K.-P. Droidmat: Android malware detection through manifest and api calls tracing. In Proceedings of the 2012 Seventh Asia Joint Conference on Information Security, Washington, DC, USA, 9–10 August 2012; pp. 62–69.
15. Wagstaff, K.; Cardie, C.; Rogers, S.; Schrödl, S. Constrained k-means clustering with background knowledge. *Icml* **2001**, *1*, 577–584.
16. Altman, N.S. An introduction to kernel and nearest-neighbor nonparametric regression. *Am. Stat.* **1992**, *46*, 175–185.
17. Wang, W.; Zhao, M.; Gao, Z.; Xu, G.; Xian, H.; Li, Y.; Zhang, X. Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions. *IEEE Access* **2019**, *7*, 67602–67631. [CrossRef]
18. Guerra-Manzanares, A.; Bahsi, H.; Nõmm, S. KronoDroid: Time-based Hybrid-featured Dataset for Effective Android Malware Detection and Characterization. *Comput. Secur.* **2021**, *110*, 102399. [CrossRef]
19. Jannath Nisha, O.S.; Mary Saira Bhanu, S. Detection of malicious Android applications using Ontology-based intelligent model in mobile cloud environment. *J. Inf. Secur. Appl.* **2021**, *58*, 102751. [CrossRef]
20. Bhandari, S.; Panihar, R.; Naval, S.; Laxmi, V.; Zemmari, A.; Gaur, M.S. Sword: Semantic aware android malware detector. *J. Inf. Secur. Appl.* **2018**, *42*, 46–56. [CrossRef]
21. Karbab, E.B.; Debbabi, M.; Derhab, A.; Mouheb, D. MalDozer: Automatic framework for android malware detection using deep learning. *Digit. Investig.* **2018**, *24*, S48–S59. [CrossRef]
22. Mathur, A.; Podila, L.M.; Kulkarni, K.; Niyaz, Q.; Javaid, A.Y. NATICUSdroid: A malware detection framework for Android using native and custom permissions. *J. Inf. Secur. Appl.* **2021**, *58*, 102696. [CrossRef]
23. Androguard: Reverse Engineering, Malware Analysis of Android Applications. Available online: https://github.com/androguard/androguard (accessed on 20 January 2021).
24. Razgallah, A.; Khoury, R.; Hallé, S.; Khanmohammadi, K. A survey of malware detection in Android apps: Recommendations and perspectives for future research. *Comput. Sci. Rev.* **2021**, *39*, 100358. [CrossRef]
25. Sihag, V.; Vardhan, M.; Singh, P. A survey of android application and malware hardening. *Comput. Sci. Rev.* **2021**, *39*, 100365. [CrossRef]
26. Zhang, W.; Luktarhan, N.; Ding, C.; Lu, B. Android Malware Detection Using TCN with Bytecode Image. *Symmetry* **2021**, *13*, 1107. [CrossRef]
27. PerDRaML. Available online: https://github.com/fahadakbar24/android-malware-detection (accessed on 8 February 2022).
28. Malware Dataset. Available online: https://github.com/fahadakbar24/android-malware-detection-dataset (accessed on 22 January 2021).
29. Chen, T.; He, T.; Benesty, M.; Khotilovich, V.; Tang, Y.; Cho, H.; Chen, K. *Xgboost: Extreme Gradient Boosting*; R Package Version 0.4-2 1; 2015; pp. 1–4. Available online: https://cran.microsoft.com/snapshot/2017-12-11/web/packages/xgboost/vignettes/xgboost.pdf (accessed on 27 October 2021).
30. Breiman, L. Random forests. *Mach. Learn.* **2001**, *45*, 5–32. [CrossRef]
31. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge, MA, USA, 2016. Available online: http://www.deeplearningbook.org (accessed on 28 October 2021).

32. Google Play Python API. Available online: https://github.com/fahadakbar24/google-play-api (accessed on 17 November 2021).
33. Sotiroudis, S.P.; Goudos, S.K.; Siakavara, K. Feature Importances: A Tool to Explain Radio Propagation and Reduce Model Complexity. *Telecom* **2020**, *1*, 114–125. [CrossRef]
34. Nasir, M.; Javed, A.R.; Tariq, M.A.; Asim, M.; Baker, T. Feature engineering and deep learning-based intrusion detection framework for securing edge IoT. *J. Supercomput.* **2022**, *78*, 1–15. [CrossRef]