


Article

# Research on Data Mining of Permission-Induced Risk for Android IoT Devices

Rajesh Kumar <sup>1,\*</sup> , Xiaosong Zhang <sup>1</sup>, Riaz Ullah Khan <sup>1</sup> and Abubakar Sharif <sup>2</sup>

<sup>1</sup> School of Computer Science & Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China; jhonsonzxs@uestc.edu.cn (X.Z.); rerukhan@gmail.com (R.U.K.)

<sup>2</sup> School of Electronic Science and & Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China; engr.abubakarsharif@gmail.com

\* Correspondence: rajakumarlohano@gmail.com; Tel.: +92-333-283-6705

Received: 25 December 2018; Accepted: 9 January 2019; Published: 14 January 2019

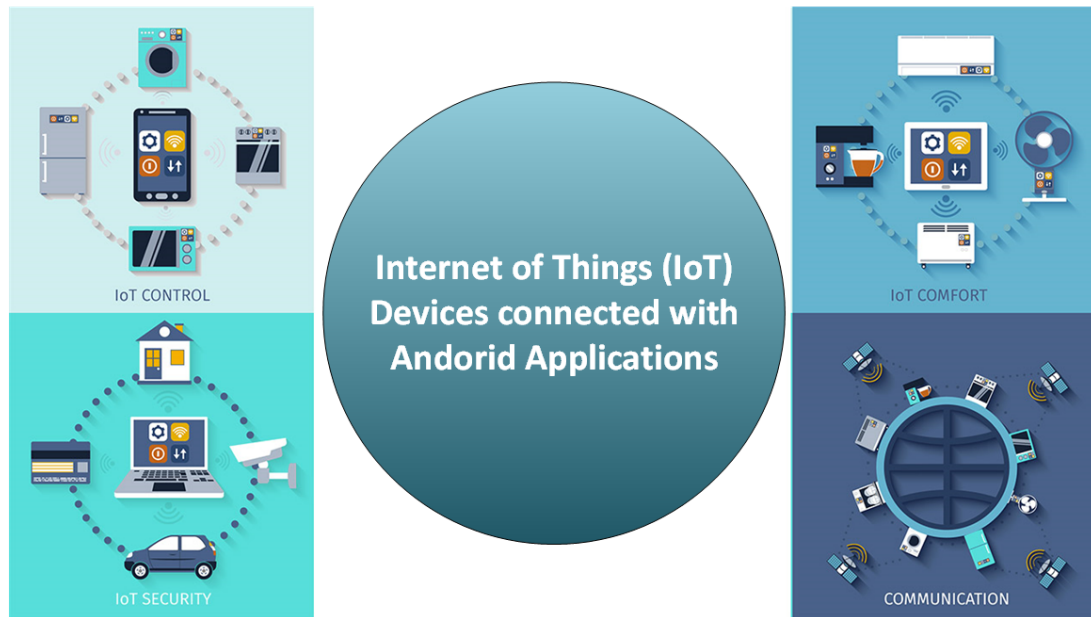


**Abstract:** With the growing era of the Internet of Things (IoT), more and more devices are connecting with the Internet using android applications to provide various services. The IoT devices are used for sensing, controlling and monitoring of different processes. Most of IoT devices use Android applications for communication and data exchange. Therefore, a secure Android permission privileged mechanism is required to increase the security of apps. According to a recent study, a malicious Android application is developed almost every 10 s. To resist this serious malware campaign, we need effective malware detection approaches to identify malware applications effectively and efficiently. Most of the studies focused on detecting malware based on static and dynamic analysis of the applications. However, to analyse the risky permission at runtime is a challenging task. In this study, first, we proposed a novel approach to distinguish between malware and benign applications based on permission ranking, similarity-based permission feature selection, and association rule for permission mining. Secondly, the proposed methodology also includes the enhancement of the random forest algorithm to improve the accuracy for malware detection. The experimental outcomes demonstrate high proficiency of the accuracy for malware detection, which is pivotal for android apps aiming for secure data exchange between IoT devices.

**Keywords:** access control; Android malware detection; computer security; permission pattern; secure machine learning

## 1. Introduction

Nowadays, smart devices have become part of the Internet of Things (IoT), and are widely used in almost every domain such as banking, shopping, social networking, etc. The adoption of the Android platform for smart devices provides a variety of services all over the world. Smart devices that use the architecture of IoT include smart TVs, refrigerators, automobiles, wrist watches, etc. Figure 1 shows some applications areas which use android apps for connecting IoT devices. In the framework of IoT, smartphones are providing numerous advantageous IoT services to clients such as cell phone's housed sensing (accelerometer, Gyroscope, compass, GPS) and connectivity alternatives (i.e., NFC, RFID, WiFi, Bluetooth). Moreover, smartphones are also used to control and monitor personal and business premises such as controlling the air temperature using Android apps or using CCTV cameras to track work progress. Consequently, in our inexorably associated society, the number and scope of Android devices keep on increasing. It is assessed that there will be roughly 6.1 billion smartphone clients by 2020 [1,2].



**Figure 1.** IoT devices Connected with Android devices and apps.

On the other side, Android devices are an increasingly attractive target for online criminals who try to hold personal details (i.e., location, contact numbers, accounts, photos, etc.) [3]. This unethical act is targeted by malicious hackers who are trying to use Android application as a tool to break into devices. In addition to this, most of the Android devices do not use anti-virus or malware detection applications [4–6]. Additionally, to control a violation of privacy and the leakage of data such as the location of the user, contact information and smartphone certificates, hackers pose a severe threat. The previous literature reports several techniques to address the problem of malware attacks, such as a sandbox, access control, signature mechanism, authority mechanism [7–9]. Furthermore, in [3], a MODroi framework based on API calls has been proposed that generate signatures which are pushed to the client devices for danger identification. However, TaintDroid [10] developed an exception-based malware discovery system based on usage of application data behaviour. Moreover, Canfora G. et al. [11], designed a structure to monitor Android Dalvik activity codes regarding frequencies, which can detect malware applications. Besides, Burguera et al. [12], proposed Crowdroid framework for the client and server components. The client uses the strace mechanism of the Linux system for monitoring android system calls. Kim et al. [13], proposed CopperDroid framework, which detects the behaviour of Java code and local code execution. Although, these reported methods are very effective, yet, these methods can not be directly applied to mobile and IoT devices, because of their limited resources such as memory and power consumption. Several techniques are developed for selecting the permission [14–17] for instance information gain technique [14], permission but they considered only high-risk permission and ignored the low-risk permissions.

Permission control is a key problem in the security of the Android operating system. Android permissions enforce the restrictions on the specific operation to offer concrete security features. However, it places a significant responsibility on app developers to declare the least privileged set of permissions required by designed apps, and it is up to the app users to completely understand the risk of granting certain permissions. The Android platform provides plenty of documentation with limited information regarding permission. The lack of reliable permission information may allow app developers to request unnecessary permissions, which may lead to overprivileged applications. Additionally, unnecessary perilous permissions may lead to malware apps, which can generate permission oriented attacks. Moreover, the lack of information about the risk level of permissions confuses the users, whether to install the app or not. Currently, the Android permission privileges are not able to help the users to make correct decisions about risk of app or security [14,17–21].

In this paper, we propose a methodology with improved accuracy for malware detection by extracting features based on Android application permissions. The behavioural analysis of permissions helps to detect malware and benign apps. Specifically, we adopted three data mining techniques in our proposed methodology (1) Permission Ranking; (2) similarity-based Permission feature selection; and (3) association rule for permission mining. The Permission Ranking analysis is used to rank the permissions based on their risk. The similarity-based Permission is used to collect the subsets of permissions (individual permissions and group of permissions), which cause a security breach in malware apps. Moreover, the association rule for permission mining discovers meaningful relationships between the permissions. Additionally, the Random Forest classifier is mostly used the algorithm in the detection of malware. Therefore, we improve the accuracy of the random forest algorithm for permission induced malware detection. The improved random forest iteratively removes the unnecessary features. By restricting the upper limit of random forest regarding the number of generated trees based on essential and unnecessary features. So, the improved random forest algorithm contains a reduced but most essential set of features. Furthermore, the experimental results validate the proposed approach for malware detection approach, which can effectively detect malware with more accuracy as compared with the previously reported techniques. Our contribution includes three major folds:

1. We develop a permission-based feature selection approach using (1) Permission Ranking, (2) similarity-based permission feature selection, for the identification of an essential subset of permissions.
2. We also evaluate the effectiveness of the mined association rules for the permission-based which improves the accuracy of prediction.
3. Finally, we enhance the performance of random forest algorithm by iteratively remove the unnecessary features and setting the upper limit on the number of trees in the random forest to improve the accuracy and recall rate, which leads to a secure data exchange between IoT devices and Android devices.

After the brief introduction of the paper, the following sections are arranged as follows. Section 2 contains the previous reported research work on Android malware detection with two major solutions. Section 3 gives a brief overview to the proposed methodology based on extracted features from application permissions. Conducted experiments and results are discussed in Section 4. The last section concludes the contribution of paper and results.

## 2. Literature Review

In this section, we briefly review the related literature work. Firstly, we discuss the static analysis, which consists of two methods (i) Permission-based analysis (ii) API Call based analysis. Secondly, we elaborate the dynamic analysis that is used to extract the training characteristics of the model. Also, we consider the hybrid analysis that combines the static and dynamic analysis. Finally, we compare the static, dynamic and hybrid analysis.

### 2.1. Static Analysis

The recent use of static features of machine learning to detect Android malware include the following: Cen et al. [22] proposed a model based on API call and permissions. This method also utilizes regularised logistic regression (RLR). Moreover, RLR was compared to different machine learning techniques such as SVM, KNN, decision tree and naive Bayes. DroidMat [23] classified the malware and benign according to the intents, permissions and API calls. For extracting static features, the author used k-nearest neighbors (k-NN) and k-means clustering algorithm. In [24], SVM classifier was developed for on-device malware detection—the proposed algorithm based on API calls, permissions and network access. Yerima et al. [9,25] used random forest ensemble learning models to detect the malware, which was based on permissions, API calls, embedded commands

and intents. Wang et al. [26] applied SVM, decision trees and random forest to analyse the use of vulnerable permissions for malware detection. Varsha et al. [27] extricate static features from the manifest and application executable documents; their location strategy gave SVM, rotation forest and random forest on three datasets. DAPASA [28] used sensitive sub-graphs to construct five features depicting invocation patterns, random forest machine learning algorithm achieved the best detection performance.

Wang et al. [26], used logistic regression, linear support vector machines, random forest and decision trees. Furthermore, the author achieved the TPR (true positive rate) of 96% and FPR (false positive rate) of 0.06% with the logistic regression, which was highest as compared with other used classifiers. The dataset used in [26], was composed of 18,363 malware applications and 217,619 benign applications, to describe the particular static signature and stage particular static features. Most of the literature explores machine learning approaches for malware detection [26,29–35]. The feature extraction methods are shown in Figure 2. Also, Table 1 describes some malware detection methods.

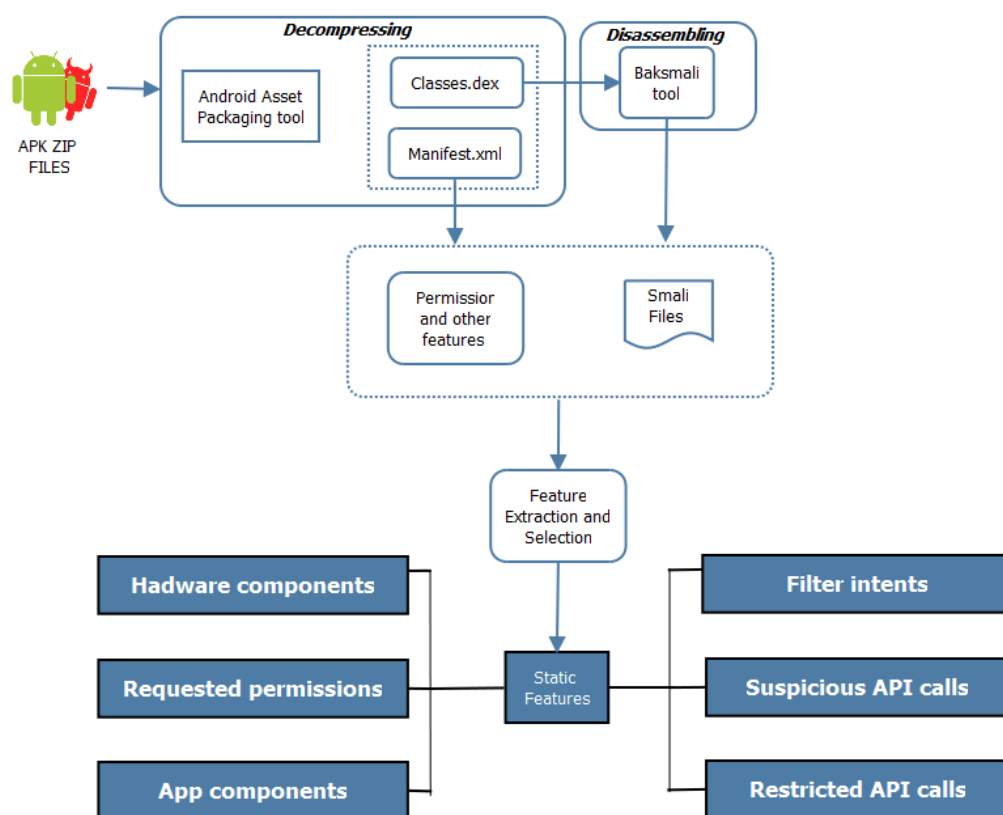


Figure 2. Feature Extraction of method.

Furthermore, Tables 1 and 2 shows the feature sets and some static features detection methods, respectively. The k-nearest neighbours machine learning classifier achieves better performance and accuracy in the detection of the malware. However, it takes more processing time with a large amount of data. That is why most of the authors used Support Vector Machine and Random Forest classifiers. Therefore, we use and enhance the Random Forest algorithm for Android malware detection.

**Table 1.** Static features detection methods.

Ref	Features	Accuracy	Machine Learning Models
[36]	Permission	91.75%	Random Forest
[19]	Permission	81%	C4.5, SVM
[37]	Permission	88.2%	HMNB
[15]	Permission	-	AHP
[21]	Permission	98.6	J48
[20]	Permission	92.79%	Random Forest
[38]	Permission	94.90%	Random Forest
[14]	Permission, API calls	92.36%	Random Forest
[23]	Permission, API calls, intent	97.87%	k-nearest neighbors
[39]	API call	99%	k-nearest neighbors
[40]	API call	93.04%	Signature matching
[41]	API call	96.69%	SVM
[42]	ICC related features	97.4%	SVM
[9]	Permission, command, API calls	98.6%	Parallel classifier

**Table 2.** Overview of feature sets.

Feature Sets	
manifest	S1 Hardware components
	S2 Requested permissions
	S3 Application components
	S4 Filtered intents
dexcode	S5 Restricted API calls
	S6 Used permission
	S7 Suspicious API calls
	S8 Network addresses

### 2.1.1. Permission-Based Analysis

Since the Android security model is based on application permissions, the permission set was extracted from the manifest file. Every application must have the privileges needed to access different features. During the application installation, the Android platform asks the user whether to grant the requested permissions. There are some permissions, which can be exploited by malicious applications. For example, a malicious application may use the permissions to access the SD card and the Internet, in order to access and filter sensitive information on an SD card. Our approach is to model the group of Android permissions requested by malicious applications. Therefore, we propose a method that uses the appearance of a specific privilege as a feature of a machine learning algorithm.

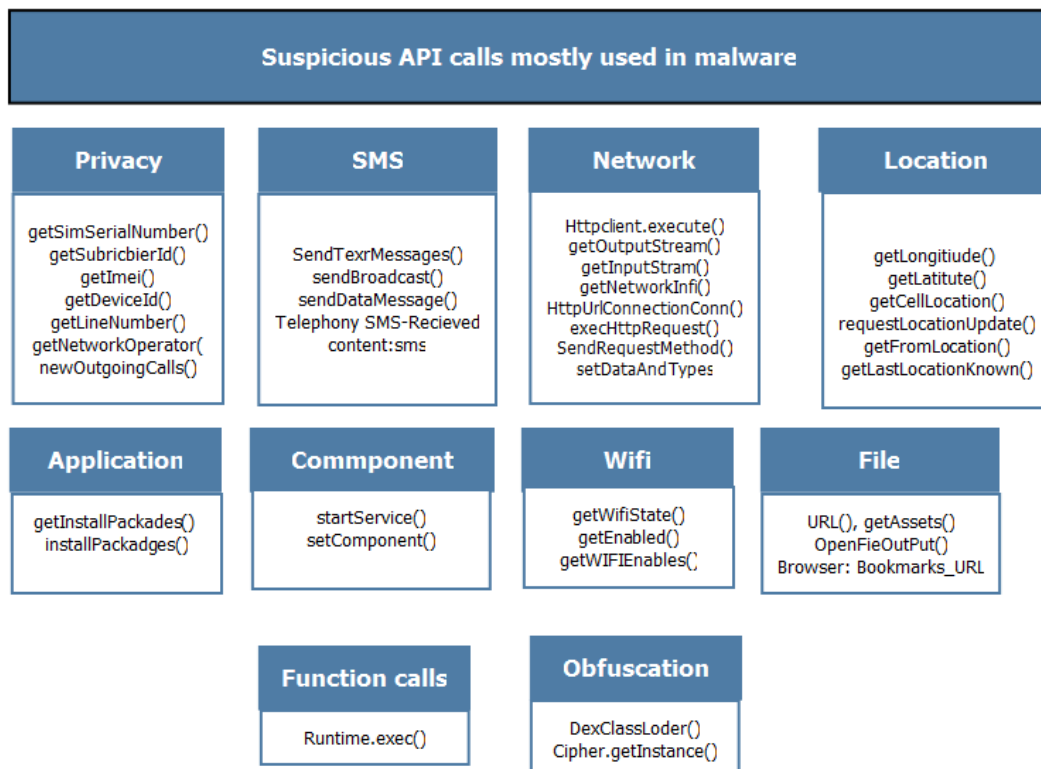
Among the 145 permission set, 48 permission are risky permissions which are mentioned in Table 3. There are several techniques for selecting the permissions [14,16,17] for instance information gain technique [14], and rank permission technique. Finally, We choose the 48 risky permission set from the previous literature [17]. In this paper, we designed the association mining rule and ranked permission methods for the detection of the malwares.

### 2.1.2. Suspicious API Calls

The second solution is a static analysis of the source code of the app. Malicious codes usually use a combination of services, methods and API calls that are not common for non-malicious applications [12]. To differentiate malicious and non-malicious applications, the Machine learning algorithms are able to learn common malware services such as combinations of APIs and system calls. Figure 3 shows the some of suspicious API calls, which are mostly used by malware applications. Figure 4 shows the extracted the features from the APK file that contains the classes.dex file.

**Table 3.** Permission set mostly used in malware.

Risky Permissions	
ACCESS_WIFI_STATE	SEND_SMS
READ_LOGS	READ_CALL_LOG
CAMERA	DISABLE_KEYGUARD
CHANGE_NETWORK_STATE	RESTART_PACKAGES
WRITE_APN_SETTINGS	SET_WALLPAPER
CHANGE_WIFI_STATE	INSTALL_PACKAGES
READ_CONTACTS	WRITE_CONTACTS
WRITE_SETTINGS	GET_TASKS
RECEIVE_MMS	ACCESS_WIFI_STATE
WRITE_APN_SETTINGS	SYSTEM_ALERT_WINDOW
READ_HISTORY_BOOKMARKS	RECEIVE_BOOT_COMPLETED
ACCESS_NETWORK_STATE	CALL_PHONE
READ_EXTERNAL_STORAGE	ACCESS_FINE_LOCATION
EXPAND_STATUS_BAR	ADD_SYSTEM_SERVICE
PERSISTENT_ACTIVITY	INTERNET
GET_ACCOUNTS	WRITE_SMS
PROCESS_OUTGOING_CALLS	CHANGE_CONFIGURATION
READ_HISTORY_BOOKMARKS	GET_PACKAGE_SIZE
WAKE_LOG	ACCESS_MOCK_LOCATION
WRITE_CALL_LOG	WRITE_HISTORY_BOOKMARKS
READ_PHONE_STATE	RECEIVE_WAP_PUSH
SET_ALARM	WRITE_SMS
RECEIVE_SMS	READ_SMS



**Figure 3.** Suspicious API calls.

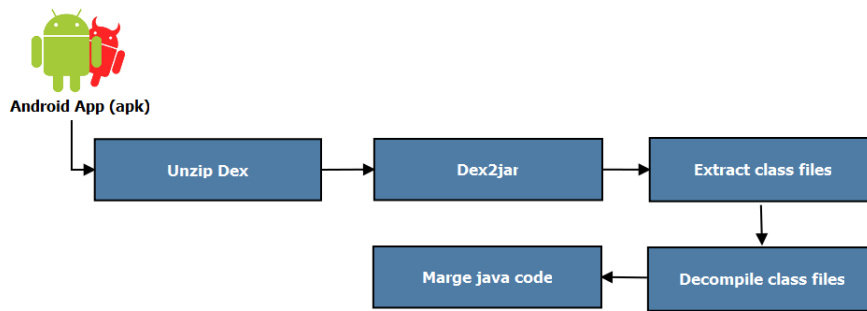


Figure 4. Workflow of Android file decompiling.

2.2. Dynamic Analysis

AntiMalDroid [43] was a machine learning method to extract dynamic features, which uses the detection technique based on behavioural sequences as feature vectors with SVM. Also, DroidDolphin [44] use Support Vector Machine with features that obtained dynamically. Afonso et al. [45] proposed a dynamic API calls and framework calls to track and study bolster vector machines, J48, IBk (an example based approach), BayesNet Pool, BayesNet K2, Random Forest, and Naive Bayes.

Figure 5 shows the feature extraction method and detection technique of the dynamic analysis. Many machine learning algorithms used for dynamic analysis for instance, Logistic regression (LR), K-means Clustering, SVM, KNN\_E, KNN, Bayesian network (BN), and Nave Bayes. Table 4 illustrates the accuracy level, dynamic features and detection methods.

Table 4. Dynamic features detection methods.

Ref	Features	Accuracy	Machine Learning Models
[46]	System call	91.75%	Signature Matching
[12]	System call	81%	K-Means
[47]	System call	88.2%	Frequency
[48]	System call	-	Pattern matching
[35]	API call	97.6	KNN_M
[19]	Native Size	99.9%	RF, SVM

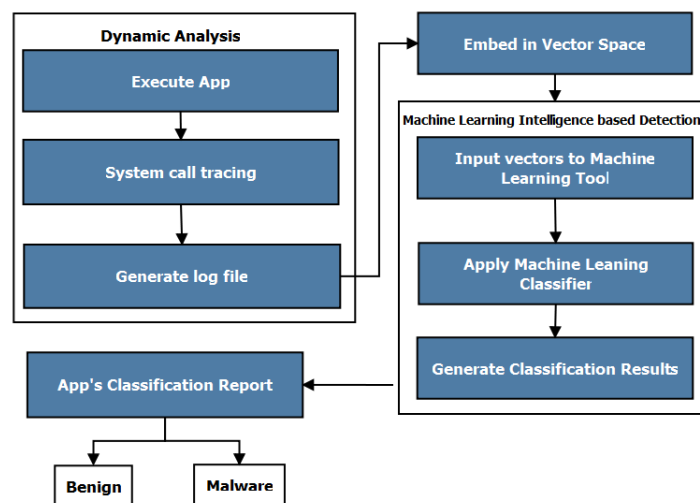


Figure 5. Dynamic Feature Extraction and Detection.



### 2.3. Hybrid Analysis

To improve the performance of learning algorithms, the hybrid analysis was developed, which uses the dynamic and static features as shown in Figure 6. Some researches proposed multi-classification techniques [49,50] to obtain high accuracy in the hybrid analysis. Furthermore, the static features are Publisher ID, API call, Class structure, Java Package name, Crypto operations, Intent receivers Services, Receivers, and Permission, and dynamic are Crypto operations, File operations, Network activity. The APK file extracted static features from classes.dex files, and dynamic features from Androidmanifest.xml file. Hybrid Analysis combines static features and dynamic features. These features are used to detect malicious applications. In [51], following features are selected from static (permission and APICall) and dynamic (SystemCall). Y. Liu, et al. [51] used the SVM and Navie Bayes machine learning classifier. The SVM classifier used for static analysis achieved 93.33 to 99.28 percent accuracy, while the Naive Bayes used for dynamic analysis achieved accuracy up to 90 percent. Furthermore, Kim et al. [13], used the J48 machine learning classifier, the features are selected from static (permission) and dynamic (APICall). A. Saracino et al. [52], achieved 96.9% accuracy based on KNN by selecting the static feature (permission) and dynamic (critical API, SMS, User activity System call) features.

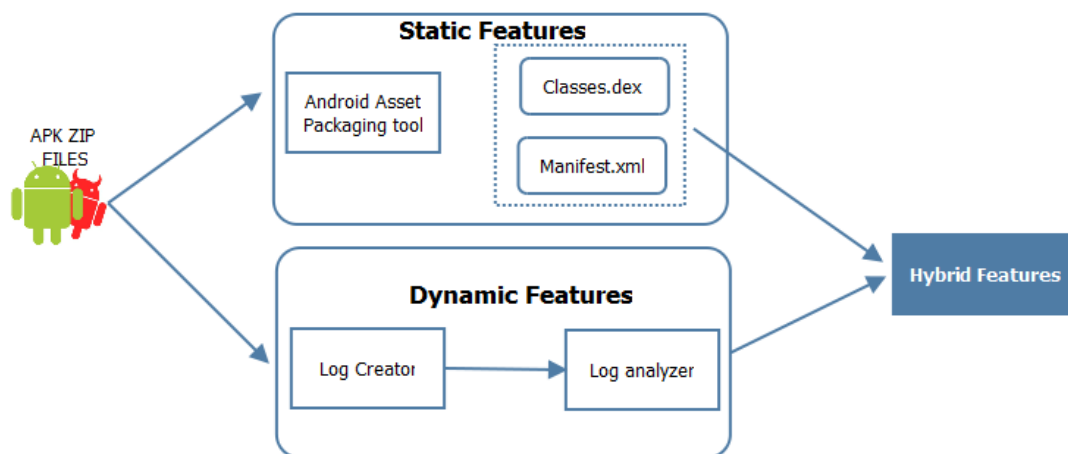


Figure 6. Dynamic Feature Extraction and Detection.

### 2.4. A Comparison of Static, Dynamic, and Hybrid Analysis

#### Static Analysis:

1. Single Category features: The advantages of single category features are easy to extract, and low power computation. The limitations associated with this method are code obstruction, imitation attack and low precision.
2. Multiple categories of Features: The advantages of multiple category features are easy to extract, and high accuracy. The limitations associated with this method are Mimicry attack, high computation, code obfuscation, and difficult to handle multiple features

#### Dynamic Analysis:

1. Single Category features: it poses a better accuracy and it is easy to recover code obfuscation as compared with static analysis. However, its feature extraction process is difficult, and it consumes high resources.
2. Multiple categories of Features: It gives better accuracy and it is easy to recover code obfuscation as compared with a static and dynamic single category. The limitations of this approach



are: (1) difficult to handle multiple features; (2) high resources; and (3) more time needed for computation.

**Hybrid Analysis:** The main benefits of hybrid analysis are to perform the highest accuracy as compared to static and dynamic analysis. The limitations are (1) highest complexity; (2) framework requirement to combine the static and dynamic features; (3) more resource use; and (4) time-consumption.

### 3. Proposed Scheme and Methodology

In this section, we propose the malware detection methods. We adopted (i) the permission ranking-based feature selection approach (ii) the similarity based permission feature selection (iii) the association rule mining algorithm and (iv) the modified random forest classifier parameters.

The permission ranking-based feature selection approach and similarity-based permission feature selection rank the features based on frequency. The association rule mining algorithm deletes the permissions, which is common in malware and benign software. Moreover, we improve the accuracy of the random forest algorithm for permission-induced malware detection. The improved random forest iteratively removes the unnecessary features. By comparing the essential and unnecessary features, we formulate an improved random forest algorithm that contains less but most imperative features.

#### 3.1. Permission Ranking

Each permission defines a specific action that the application is permitted to perform. For example, the permission INTERNET specifies the user can access the Internet. Different kinds of benign and malicious applications can request various permissions that correspond to their operational requests. For developing real-time Android malware detection system do not analyze all permission for the malicious application, it needs some common features of the permissions [17].

However, we pay more attention to creating permissions on high—risk external attacks and are often requested by malware samples. Therefore, malware examples rarely require permission to the good indicator of the distinction among malicious and benign applications. Thus, our methodology classifies the highly distinguishable permissions so that we can use the information for classify the malicious apps and benign apps. In addition, we exclude common permission which is used in benign and malicious application because they create ambiguity to detect the malware. For example, both malware and benign applications often request INTERNET permissions because almost all applications require access to the Internet. So our method identifies INTERNET permission.

We demonstrate the schemes which analyze the permission ranking that can be utilized to distinguish malware from benign and malicious application. Ranking is definitely not a new concept. Wang et al. [17], used the ranking-based method, but it only identifies the high-risk permissions. Previous work ignored low-risk permissions because they were interested in identifying malware abuse, and our goal is to develop ranking-based framework that can detect malicious and benign apps. In addition, in the Android platform each permission has a different operation. For instance, permission READ\_SD\_CARD indicates that apps has access to the mobile disk. We are focusing on the permission to create high risk of the attack by the malware apps. we are focusing to classify the malicious and benign apps for the detection of malware.

The method is based on two matrices:  $M$  specify the malware application permission list and  $B_{ij}$  indicate the list of permissions for the benign application.

Before calculating the permission from matrices B and M, We check the size of benign and malicious applications. For instance, the quantity of benign application is much larger then malicious application due to this reason dataset is imbalanced. An imbalanced dataset can cause you a lot of frustration. If the size of benign B app is larger than malware M apps for the balance of the two matrices mentioned in the following equation

$$S_B(P_j) = \frac{\sum_i B_{ij}}{\text{size}(B_j)} * \text{size}(M_j)$$

where  $P_j$  represented the  $j$ th permission and  $SB(P_j)$  denotes the  $j$ th permission in matrix

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

In the equation above  $R(P_j)$  denotes the rate of the  $j$ th permission. The value if  $R(P_j)$  between  $[-1, 1]$ . If the value of  $R(P_j) = 1$ , that means  $(P_j)$  is malicious if  $R(P_j) = -1$  that means benign permission. As we know malicious permission High-Risk permissions and benign low-risk permission.  $R(P_j) = 0$ , impact on  $(P_j)$  is is much less effective in malware detection. We generate two kinds of lists, one in ascending order and other in descending order. Next we recognize the frequency of top permission list used in benign and malicious applications.

### 3.2. Similarity-Based Permission Feature Selection

In this section, we matched the permission list  $P.P^{sb} = p_1, p_2, p_3, \dots, p_n$  corresponding to the marking function  $\gamma$  the similarity of permission shown in the equation

$$S_B(P_j) = e_p \sum_{i=1}^n \gamma^{S_B} (P_j^{sb}) \psi (P_j^{sb}), (P_j) \in P^{sb}$$

Thus, we can calculate the similarity between features

$$S_{score} = S_p + S_j$$

We calculate the maximum similarity of the permission lists and observed the threshold. If threshold is exceed than we consider as a malware otherwise benign. In our experiment threshold and support different value is 0.05 and 0.15 respectively.

### 3.3. Association Rule Mining Algorithm Based on Probabilistic Model

Association rule mining is used to discover meaningful relationships between variables in huge databases. For example, if events A and B always occur at the same time, then the two events are likely to be associated. for instance, we found that many permission are always together i.e., READ\_CONTACTS and WRITE CONTACTS are always used together. These dangerous Android permissions belong to Google’s permission list. As we know, those permissions are always together. So we only need one of them to characterize certain behavior. We need to remove the higher association permission i.e., READ\_CONTACTS. In this paper, we define association rules as those with low support but high confidence and proposed association rule mining algorithm for finding the permissions that occur together.

- STEP1: Find out the frequent two-permissions sets
- STEP2: Diversity-based interestingness measures for association rule using frequent two itemsets that was developed by Piattetsky- Shapiro [53]
- When  $\text{support}(Y \cup Z) \approx \text{support}(Y)\text{support}(Z)$ , the two-item sets  $(Y, Z)$  are mutually independent. That is, the association rule  $Y \Rightarrow Z$  is uninteresting.

$$\begin{aligned} \text{interest}(y, z) &= \frac{\text{support}(Y \cup Z)}{\text{support}(Y)\text{support}(Z)} - 1 \\ &= \frac{P(Y|Z)}{P(Z)} \end{aligned}$$

- if  $\text{interest}(Y, Z) > 0$ ,  $Y$  and  $Z$  are correlated positively.

- if  $\text{interest}(Y, Z) \approx 0$ ,  $Y$  and  $Z$  are commonly independent, and the common two-item sets should be rejected.
  - if  $\text{interest}(Y, Z) < 0$ ,  $Y$  and  $Z$  are negatively correlated.
- STEP3: Create the association rule based on the permission shown in Algorithm 1
- STEP4: Calculate probability table of the association rules.

---

**Algorithm 1** Association Rule set R For Permission Based
 

---

```

1: input  $\leftarrow$  1 Association Rule Set R
2: minSub  $\leftarrow$  minimum threshold of support coefficient
3: minConf  $\leftarrow$  minimum threshold of confidence coefficient
4: for Z=D do
5:   r = null
6:   r.PushTail(Z)
7:   for Y in D do
8:     if  $Y \Rightarrow Z \in L2$  and  $\text{support}(Y \Rightarrow Z) > \text{minsup}$  and  $\text{confidence}(Y \Rightarrow Z) > \text{minconf}$  then
9:       r.PushTail(Y)
10:    end if
11:    r.PushTail(r)
12:  end for
13: end for
14: output  $\leftarrow$  Association Rule R
  
```

---

### 3.4. Improved Random Forest Classifier

The voting decision process is an important part of the RF algorithm and it determines the final classification of the test sample. The RF algorithm adopts the principle of simple voting. Each decision tree is given the same weight, ignoring the difference between the strong classifier and the weak classifier, and affecting the overall classification performance of the random forest classifier. For this reason, this paper adopts the weighted voting principle to modify the RF algorithm to form an improved IRF (improved random forest) Algorithm 2. The Table 5 shows the list of symbols used in Algorithm 2. The architecture of training and testing shown in Figure 7 and description of the algorithm is shown in Figure 8. we have changed the few parameters of the random forest algorithm as we can see in line no. 11 and 12, we tuned the parameters. The line no. 11, iteratively removes the unnecessary features. By comparing the essential and unnecessary features, we formulate an improved random forest algorithm that contains less but most important features. Consequently, we enhance the accuracy of the random forest algorithm by removing the extra features and further improves the more performance in feature selection.

---

**Algorithm 2** Modified Random Forest (IRF)
 

---

```

1: Grow initial forest ( $\theta_0$ ) and  $B_0$  random trees and feature vector  $F_{0(.)}$ 
2: An average ranking calculated weight  $w(.)$  ranked the all features  $F_{0(.)}$ 
3: Features from the ranked list, place the top  $u_0 = \sqrt{F_{0(.)}}$  in  $T_0$ 
4: Put rest  $V_0 = \#F_{0(.)} - u_0$  features in  $t_1$ 
5: n is the number of pass. Initialize  $n = 0$ 
6: for  $u_n > f$  do
7:   compute mean  $u_n$  and standard deviation  $\sigma_n$  of features weights in  $t_1$ 
8:   Find  $R_n = \forall j \in t_1 : w(j) > (u_n - 2\sigma_n)$ , if no such j exist  $R_n = \{j\}, j \in t_1 : w(j) = \min W(k)$ 
9:   get rid of unimportant features, find the most informative feature set  $A_n$  whose weight greater than the minimum value of the important features weight. so  $A_n = \{j\} : w(j) = \min \forall j \in t_{n+1}, k \in t_{n+1}$ 
10:  Find  $F_{n+1}(.) = F_n(.) - R_n$ 
11:  Find  $T_{n+1} = T_n + A_n$  and  $T_{n+1}^1 = T_n^1 - A_n$ 
12:   $u_{n+1} = \#T_{n+1}$  and  $u_{n+1} = \#T_{n+1}^1$ . Calculate  $\Delta u_n = u_{n+1} - u_n$  and  $\Delta v_n = v_{n+1} - v_n$ 
13:  Find  $|v\Delta B| < lq_u \Delta u + lq_v \Delta v; B_{n+1} = B_n + \Delta B$ 
14:  Grow forest ( $\theta_0$ ) and  $B_0$  trees and feature vector  $F_{n+1}(.)$ 
15:  Calculate Weights  $w(.)$  and ranked the all features  $F_{n+1}(.)$ 
16:   $n = n + 1$ 
17: end for
  
```

---

Table 5. List of Symbols.

Symbol	Definition
$F_{0(.)}$	Initial feature vector
$F_{n(.)}$	Feature vector at the $n^{th}$ construction process
$u_n$	Number of important features
$v_n$	Number of unimportant features
$\theta_n$	Forest at the $n^{th}$ construction process
$B_n$	Number of trees
$T_n$	Bag of important features
$T_n^1$	Bag of unimportant features
$w(j)$	Weight of features $j$
$R_n$	remove features until all the features have been eliminated
$A_n$	new feature “mark as important”
$n$	Classification accuracy

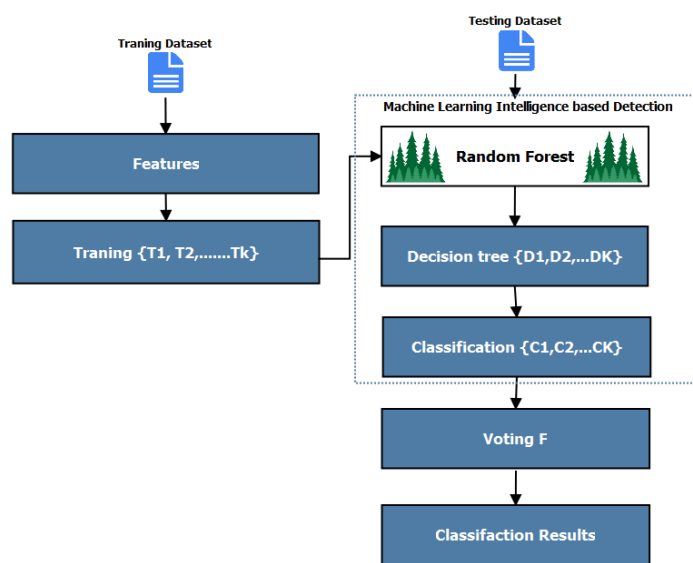


Figure 7. Random Forest Algorithm.

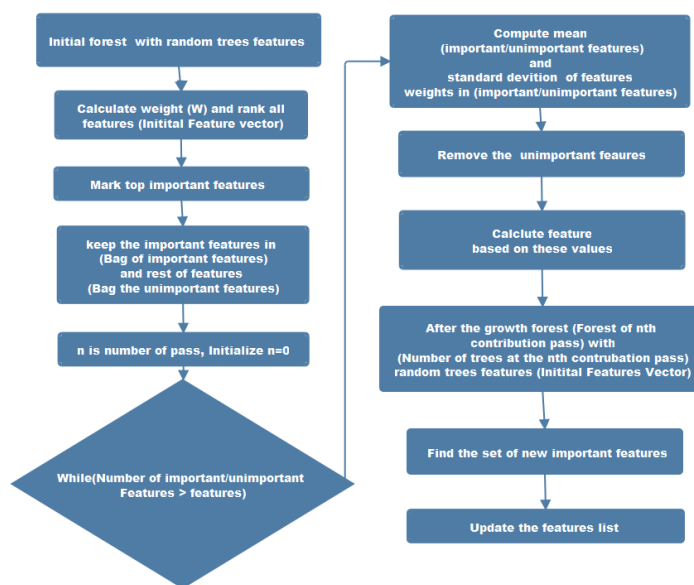


Figure 8. Flow Chart of Random Forest Algorithm.

#### 4. Experimentation and Results

To calculate the effectiveness of the machine learning classifier such as Random Forest, J48 and Naive Bayes. we select the formulas to evaluate the classifiers and effectiveness of the dataset.

TN True negative rate—the rate of malware detection recognized correctly as malicious.

TP True positive rate—the rate of benign apps detection recognized incorrectly as benign.

FP False positive rate—the rate of malware recognized incorrectly as malicious.

FN False negative rate—the rate of benign apps recognized incorrectly as benign.

Given the number of true positives and false negatives, recall is calculated using the following formula

$$TPR = \frac{TN}{TN + FP}$$

The TPR is sometimes referred to as “sensitivity” or the “true positive rate”. Given the number of true positive and false positive classified items, precision (also known as “positive predictive rate”) is calculated as follows:

$$FPR = \frac{FN}{FN + TP}$$

##### 4.1. DATASET

To excavate practical significance, it can cover most Android permission models, which have their own characteristics. In this paper, our dataset composed of malware and benign application. The dataset includes 6192 benign and 5560 malware apps, collected from the Google Play Store and the Chinese App store. Figure 9 shows the frequency of the permission. Therefore, we collect dataset from multiple sources for considering the more attributes.

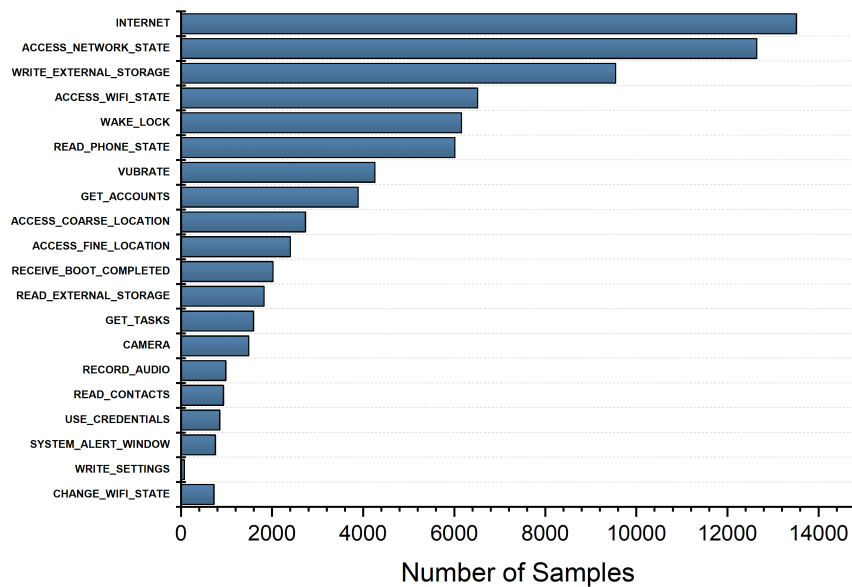


Figure 9. Top 20 Permission Frequency in dataset.

##### 4.2. Ranking and Similarity Based Frequency of the Permissions

We generate the top 10 permission combination based on probability and negative rate. The frequency of benign and malware application with k = 1, 2, 3, 4 are shown in Figures 10–13, respectively. When k = 1, READ\_SMS, SEND\_SMS, WRITE\_SMS and RECIEVE\_SMS much more commonly than the benign applications. To detect these two types of important permissions k = 2, then two permission more frequently appear in malware (READ PHONE STATE & READ SMS, INTERNET & READ SMS). Through the combination of INTERNET and READ SMS permissions,

malware can send private messages over the Internet; through a combination of READ PHONE STATE and READ SMS permissions, malware can read phone IDs such as IMSI (International Mobile Subscriber Identification) and IMEI (International Mobile Equipment Identity), as well as SMS messages, which can be used to detect mobile users and collect personal information. For larger  $k$ , the combination of permissions obtained is more intrusive. We found that the larger the value of  $k$ , the higher the precision of the permission combination to identify malware. We studied the results of  $k = 1, 2, 3, 4$ , and found that some permission are similar malicious and benign applications. While some combinations of permissions are clearly malware are requested more frequently than benign applications. Almost all combinations of permissions frequently requested by malware include SMS-related permissions, such as READ\_SMS and WRITE\_SMS. We suspect this is because most of the examples in malware datasets are related to SMS attacks, such as intercepting and leaking SMS messages, and sending SMS messages to advanced numbers. The top 10 patterns according to common and run time permission pattern shows in Table 6.

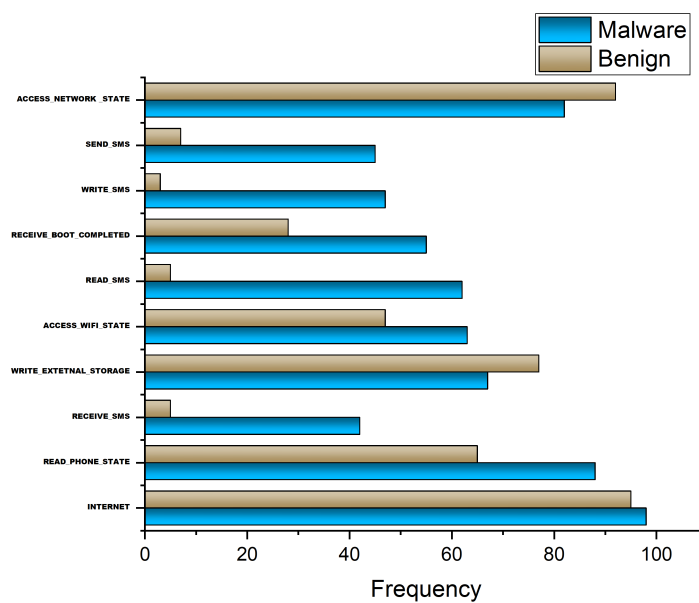


Figure 10. Top 10 Permission Combination  $k = 1$ .

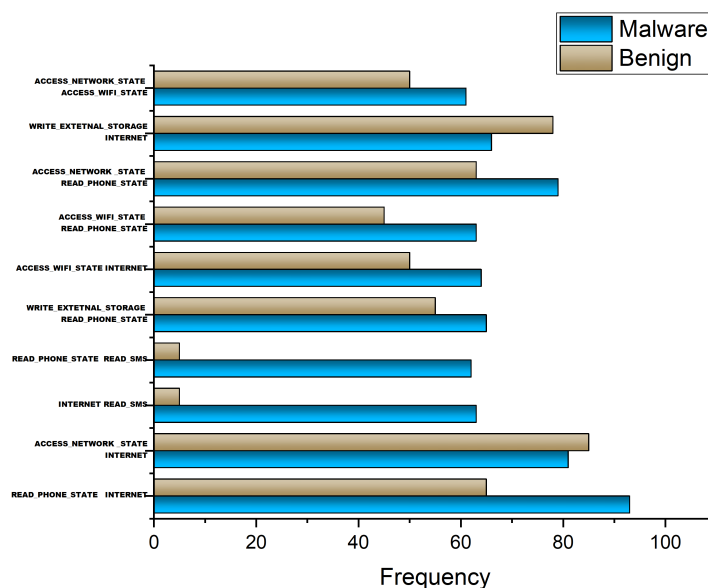


Figure 11. Top 10 Permission Combination  $k = 2$ .

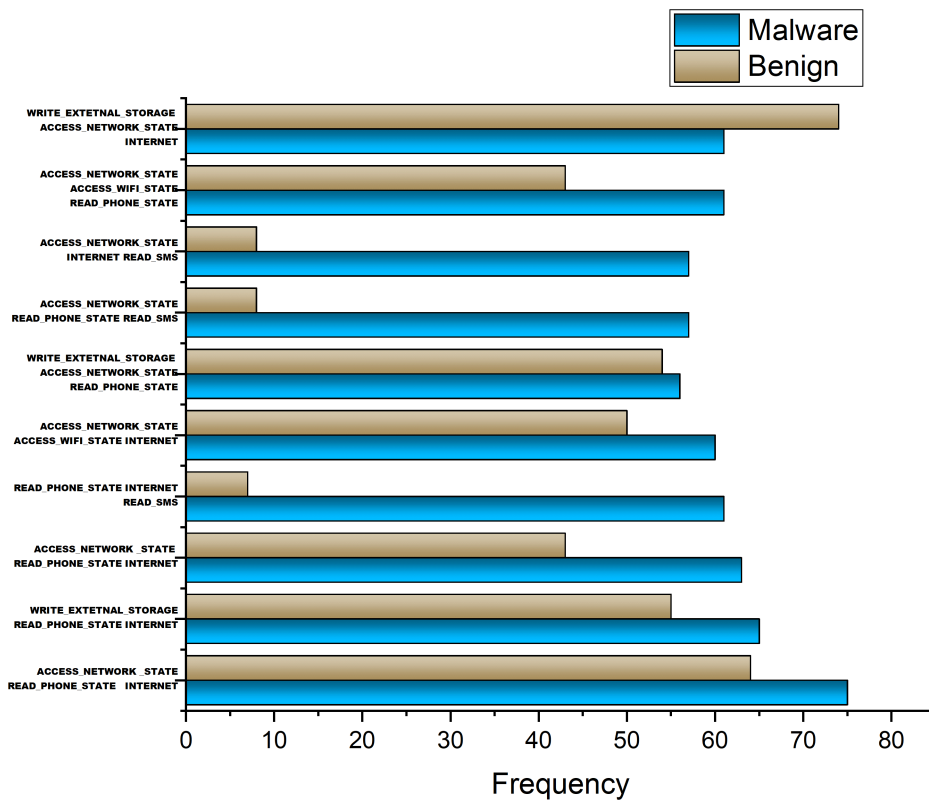


Figure 12. Top 10 Permission Combination k = 3.

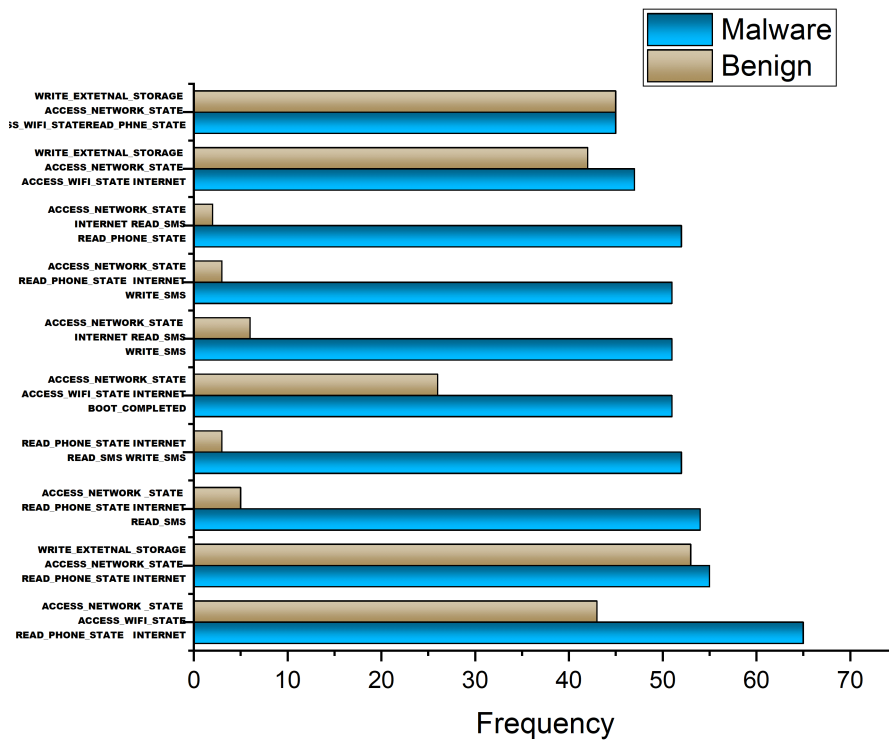


Figure 13. Top 10 Permission Combination k = 4.



**Table 6.** Permission Patterns Malware and Benign.

Permission Patterns	Benign	Malware
Common Android request permission		
READ_PHONE_STATE, ACCESS_WIFI_STATE	2.36	63.08
INTERNET, ACCESS_WIFI_STATE	5.05	63.49
READ_PHONE_STATE	31.87	93.4
ACCESS_WIFI_STATE	5.22	63.49
ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE	3.99	60.31
INTERNET, WRITE_EXTERNAL_STORAGE, READ_PHONE_STATE	13.28	65.44
INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE	24.21	78.97
INTERNET, READ_PHONE_STATE	31.21	93.078
WRITE_EXTERNAL_STORAGE, READ_PHONE_STATE	13.37	65.53
READ_PHONE_STATE, ACCESS_NETWORK_STATE	24.21	79.05
Common Android Run-time Permissions		
READ_PHONE_STATE, ACCESS_NETWORK_STATE	23.63	77.18
INTERNET, READ_LOGS	6.85	6.85
READ_PHONE_STATE	30.32	91.69
INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE	26.36	77.18
READ_PHONE_STATE, VIBRATE	21.92	65.28
INTERNET, READ_PHONE_STATE	29.9	91.52
READ_PHONE_STATE, READ_LOGS	5.38	46.86
READ_LOGS	6.93	47.6
INTERNET, READ_PHONE_STATE, VIBRATE	21.68	65.12
Unique Android request permission		
READ_PHONE_STATE, WRITE_SMS	0	50.94
INTERNET, READ_PHONE_STATE, ACCESS_WIFI_STATE	0	63.09
ACCESS_NETWORK_STATE, RECEIVE_BOOT_COMPLETED	0	51.68
ACCESS_NETWORK_STATE, WRITE_SMS	0	49.64
RECEIVE_BOOT_COMPLETED, ACCESS_WIFI_STATE	0	42.63
INTERNET, RECEIVE_BOOT_COMPLETED	0	44.75
WRITE_EXTERNAL_STORAGE, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE	0	54.53
READ_PHONE_STATE, RECEIVE_BOOT_COMPLETED	0	43.12
INTERNET, SEND_SMS	0	43.12
INTERNET, ACCESS_NETWORK_STATE, ACCESS_WIFI_STATE	0	60.31
Unique Android Runtime Permissions		
INTERNET, READ_PHONE_STATE, ACCESS_NETWORK_STATE, VIBRATE	0	55.42
ACCESS_NETWORK_STATE, VIBRATE, READ_LOGS	0	38.55
READ_PHONE_STATE, ACCESS_NETWORK_STATE, READ_LOGS	0	43.2
READ_LOGS, INTERNET, ACCESS_NETWORK_STATE	0	43.2
READ_PHONE_STATE, VIBRATE, READ_LOGS	0	41.33
INTERNET, VIBRATE, READ_LOGS	0	41.49
READ_LOGS, INTERNET, READ_PHONE_STATE,	0	46.87
ACCESS_FINE_LOCATION, READ_PHONE_STATE, VIBRATE, INTERNET	0	34.23
INTERNET, SEND_SMS	0	33.58
INTERNET, ACCESS_FINE_LOCATION, READ_LOGS	0	28.45

#### 4.3. Association Rule Mining Algorithm Based Feature Selection

We generate the top permission based on data mining association rule Algorithm. The permission WRITE\_SMS frequently appear in benign and malware, while READ\_SMS is mainly used by malware. When we delete the privilege WRITE\_SMS, the application requesting permission READ\_SMS is likely to be classified as malware. Furthermore, we found that WAKE\_LOCK, and READ\_HISTORY\_BOOKMARKS and WRITE\_HISTORY\_BOOKMARKS, and READ\_PHONE\_STATE have a high occurrence of being associated. After trimming the three permission characteristics in the dataset, we only reserved some features. Then we observed that a new model using the permissions set in Table 7.

**Table 7.** Random Forest Based Malware Detection for Permissions.

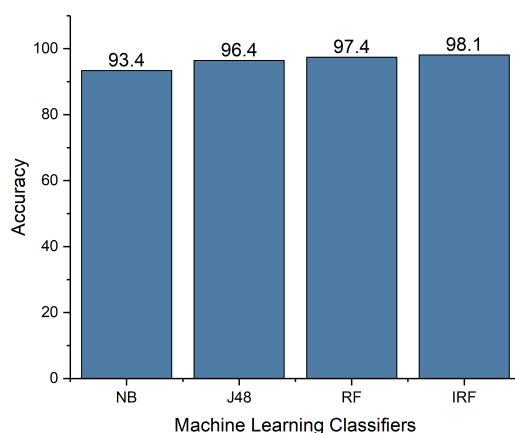
Random Forest Based Malware Detection for Permissions	
ACCESS_WIFI_STATE	SEND_SMS
READ_LOGS	READ_CALL_LOG
RESTART_PACKAGES	DISABLE_KEYGUARD
READ_EXTERNAL_STORAGE	CHANGE_NETWORK_STATE
WRITE_APN_SETTINGS	SET_WALLPAPER
CHANGE_WIFI_STATE	INSTALL_PACKAGES
READ_CONTACTS	WRITE_CONTACTS
CAMERA	GET_TASKS
READ_HISTORY_BOOKMARKS	ACCESS_WIFI_STATE
WRITE_APN_SETTINGS	SYSTEM_ALERT_WINDOW
WRITE_SETTINGS	RECEIVE_BOOT_COMPLETED

We arrange that different permission ordering methods result in different sorted lists. For instance, we give up INTERNET permissions because it indicates that both benign and malicious applications usually require INTERNET. However, the mutual information-based ranking method saves the permission INTERNET, because all applications often request the permission INTERNET. Therefore, we believe that our algorithms can preserve more important permissions by deleting unimportant or irrelevant permissions. Our approach distinguish malware applications (with a higher review rate), which is necessary for malware detection.

#### 4.4. Machine Learning Malware Detection

Our experiment uses different kinds of machine learning algorithms (i.e., naive Bayes, Random Forest iterative Random Forests and J48 decision tree). We used machine learning techniques after using the probability-based model and association-based algorithm. We evaluate the performance of the random forest algorithm regarding detection accuracy. Figure 14 shows the machine learning models; the random forest algorithm achieves more than 98% accuracy, we achieve the highest recall rate for detection the dangerous permission of Android platform. Table 8 shows the improved random forest achieved the best performance and provided the highest efficiency and lowest false alarm rate.

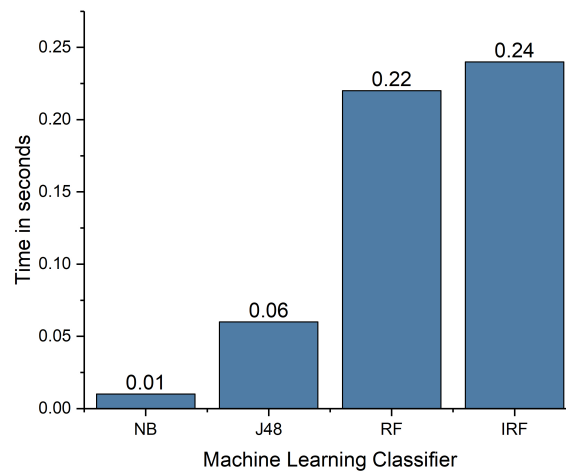
In Figure 15, we obtained the average processing time of the random forest, naive Bayes and J48 machine learning classifier based on the recall rates of the permissions. As shown, naive Bayes is the most less time-consuming machine learning algorithm. When using a naive Bayes with 10 significant permissions, the processing time averages only 0.1 s, compared to the improved random forest 0.24 s for the malware detection. In addition, our approach report that recall rate for detection the dangerous permission of Android platform shown in Figure 16.



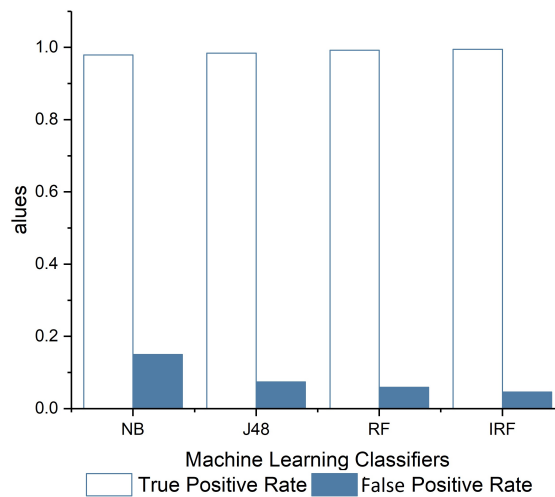
**Figure 14.** Performance and comparison of machine learning classifiers.

**Table 8.** Comparison different machine learning classifiers.

Alogrthim	TP	FP	TN	FN	TPR	FPR	ACC
NB	1233	101	573	27	0.979	0.150	0.934
J48	1240	50	624	20	0.984	0.074	0.964
RF	1250	40	634	10	0.992	0.059	0.974
IRF	1254	31	643	6	0.995	0.046	0.981



**Figure 15.** Modeling time comparison of different classifiers.



**Figure 16.** Comparison between different machine learning classifiers on dynamic features.

#### 4.5. Compared with Other Methods

There are several techniques developed for selecting the permission [14,16,17] for instance information gain technique [14], and rank permission technique. Based on these permission set, we designed the association mining rule and ranked permission methods for detecting the malware.

We analyse our results by comparing them with the other methods proposed by Wang et al. [17], which uses the ranking-based approach, but it only identifies the high-risk permissions and ignores the low-risk permissions. We focused on the low-risk as well as high-risk permissions. We additionally observe that, despite only a small amount of permissions, our methodology is as yet better than most existing malware scanners currently available. Some detection techniques rely only on signature,

it looks for specific patterns, so if a particular type of malware signature pattern is not matched, then the system will not be able to detect the particular type of malware.

The DREBIN [24] method was used for static analysis to build datasets based on application permissions and other features. Our approach is more efficient than DREBIN when combining permission. Also, the support vector machine (SVM) algorithm was used to classify malware datasets. It is a little challenging to enhance DREBIN and SVM. Previous research [14,20,36,38] shows the Random Forest achieved the best performance in detection of malware application.

Therefore, we modify the Random Forest algorithm to achieve better accuracy as compared with a simple Random Forest algorithm. The result shows that the proposed malware detection approach can effectively detect malware with more accuracy (98.1%). Moreover, the true positive rate and false positive rate of improved random forest are, 95.5% and 4.6%, respectively. The experimental results show improved random forest algorithm is effective for the detection of malware. Besides the significance of proposed approach towards malware detection, the limitations associated with this approach include accuracy comparison with other studies and handling towards the feature hiding techniques when decompile the apk using Dex2jar. Furthermore, in future, we plan to develop a framework based on the blockchain to combine static and dynamic analysis for run-time malware detection.

## 5. Conclusions

Recently, there is an increase in the number of IoT devices connected to the Internet. Most of these IoT devices use Android applications for communication and data exchange. The permission mechanism of the Android platform restricted the access of the applications. Permission can be used as elements of Android applications to detect benign apps and malware apps. However, our work reduced the number of permission for maintaining accuracy and high effectiveness. In this work, Firstly, we adopted three data mining techniques (1) Permission Ranking; (2) similarity-based Permission feature selection; and (3) association rule for permission mining. The permission ranking analysis and similarity-based permission are used to rank the permissions based on their risk and to collect the subsets of permissions (individual permissions and group of permissions), respectively. Moreover, the association rule for permission mining discovers meaningful relationships between the permissions. Secondly, we improve the accuracy of the random forest algorithm for permission induced malware detection. The accuracy of random forest was improved by modifying selective parameters of the algorithm, (1) iteratively removing the unnecessary features, (2) by setting the upper limit on the number of trees in the random forest. The result shows that the proposed malware detection approach can effectively detect malware with more accuracy (98.1%) with true positive rate and false positive rate, 95.5% and 4.6%. Furthermore, the development of a framework based on the blockchain to combine static and dynamic analysis approaches for run-time malware detection would be a good topic for future work.

**Author Contributions:** Formal analysis, R.K.; Methodology, R.K.; Project administration, X.Z.; Validation, R.U.K.; Writing—original draft, R.K.; Writing—review and editing, A.S.

**Acknowledgments:** This research was supported by the National Natural Science Foundation of China under grant No. 61572115.

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

## References

1. Park, J.S.; Youn, T.Y.; Kim, H.B.; Rhee, K.H.; Shin, S.U. Smart contract-based review system for an IoT data marketplace. *Sensors* **2018**, *18*, 3577. [[CrossRef](#)] [[PubMed](#)]
2. Risteska Stojkoska, B.L.; Trivodaliev, K.V. A review of Internet of Things for smart home: Challenges and solutions. *J. Clean. Prod.* **2017**, *140*, 1454–1464. [[CrossRef](#)]

3. Damshenas, M.; Dehghantanha, A.; Choo, K.K.R.; Mahmud, R. M0Droid: An Android Behavioral-Based Malware Detection Model. *J. Inf. Priv. Secur.* **2015**, *11*, 141–157. [[CrossRef](#)]
4. Walls, J.; Choo, K.K.R. A review of free cloud-based anti-malware apps for android. In Proceedings of the 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, Helsinki, Finland, 20–22 August 2015; Volume 1, pp. 1053–1058. [[CrossRef](#)]
5. Chen, H.; Su, J.; Qiao, L.; Xin, Q.; Chen, H.; Su, J.; Qiao, L.; Xin, Q. Malware Collusion Attack against SVM: Issues and Countermeasures. *Appl. Sci.* **2018**, *8*, 1718. [[CrossRef](#)]
6. Dogru, B.; Kiraz, O. Web-Based Android Malicious Software Detection and Classification System. *Appl. Sci.* **2018**, *8*, 1622. [[CrossRef](#)]
7. Sui, L. Strategy Analytics: Android captures record 88 percent share of global smartphone shipments in Q3 2016. *Strateg. Anal. Res. Experts Anal.* **2016**, *28*, 28–35.
8. Demontis, A.; Melis, M.; Biggio, B.; Maiorca, D.; Arp, D.; Rieck, K.; Corona, I.; Giacinto, G.; Roli, F. Yes, Machine Learning Can Be More Secure! A Case Study on Android Malware Detection. *IEEE Trans. Dependable Secur. Comput.* **2017**, 5971. [[CrossRef](#)]
9. Yerima, S.Y.; Sezer, S.; Muttik, I. Android malware detection using parallel machine learning classifiers. In Proceedings of the 2014 8th International Conference on Next Generation Mobile Applications, Services and Technologies, NGMAST 2014, Oxford, UK, 10–12 September 2014. [[CrossRef](#)]
10. Enck, W.; Gilbert, P.; Chun, B.G.; Cox, L.P.; Jung, J.; McDaniel, P.; Sheth, A.N. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. *Commun. ACM* **2014**, *57*, 99–106. [[CrossRef](#)]
11. Canfora, G.; Mercaldo, F.; Visaggio, C. Mobile malware detection using op-code frequency histograms. In Proceedings of the SECRYPT 2015—12th International Conference on Security and Cryptography, Part of 12th International Joint Conference on e-Business and Telecommunications (ICETE 2015), Colmar, France, 20–22 July 2015. [[CrossRef](#)]
12. Burguera, I.; Zurutuza, U.; Nadjm-Tehrani, S. Crowdroid: Behavior-Based Malware Detection System for Android. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices—SPSM '11, Chicago, IL, USA, 17–21 October 2011; p. 15. [[CrossRef](#)]
13. Kim, J.; Choi, H.; Namkung, H.; Choi, W.; Choi, B.; Hong, H.; Kim, Y.; Lee, J.; Han, D. Enabling Automatic Protocol Behavior Analysis for Android Applications. In Proceedings of the CoNEXT 2016—Proceedings of the 12th International Conference on Emerging Networking EXperiments and Technologies, Irvine, CA, USA, 12–15 December 2016; pp. 281–295. [[CrossRef](#)]
14. Chan, P.P.; Song, W.K. Static detection of Android malware by using permissions and API calls. In Proceedings of the International Conference on Machine Learning and Cybernetics, Lanzhou, China, 13–16 July 2014; Volume 1, pp. 82–87. [[CrossRef](#)]
15. Dini, G.; Martinelli, F.; Matteucci, I.; Petrocchi, M.; Saracino, A.; Sgandurra, D. Risk analysis of Android applications: A user-centric solution. *Future Gener. Comput. Syst.* **2018**. [[CrossRef](#)]
16. Seo, S.H.; Gupta, A.; Sallam, A.M.; Bertino, E.; Yim, K. Detecting mobile malware threats to homeland security through static analysis. *J. Netw. Comput. Appl.* **2014**, *38*, 43–53. [[CrossRef](#)]
17. Wang, W.; Wang, X.; Feng, D.; Liu, J.; Han, Z.; Zhang, X. Exploring permission-induced risk in android applications for malicious application detection. *IEEE Trans. Inf. Forensics Secur.* **2014**, *9*, 1869–1882. [[CrossRef](#)]
18. Felt, A.; Chin, E.; Hanna, S. Android permissions demystified. In Proceedings of the 18th ACM Conference on Computer and Communications Security—CCS '11 (2011), Chicago, IL, USA, 17–21 October 2011; pp. 627–636. [[CrossRef](#)]
19. Huang, C.Y.; Tsai, Y.T.; Hsu, C.H. Performance Evaluation on Permission-Based Detection for Android Malware. *Smart Innov. Syst. Technol.* **2013**, *12*. [[CrossRef](#)]
20. Kumar, A.; Kuppusamy, K.S.; Aghila, G. FAMOUS: Forensic Analysis of MOBILE devices Using Scoring of application permissions. *Future Gener. Comput. Syst.* **2018**. [[CrossRef](#)]
21. 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**. [[CrossRef](#)]
22. Cen, L.; Gates, C.S.; Si, L.; Li, N. A Probabilistic Discriminative Model for Android Malware Detection with Decompiled Source Code. *IEEE Trans. Dependable Secur. Comput.* **2015**, *12*, 400–412. [[CrossRef](#)]

23. 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 7th Asia Joint Conference on Information Security (Asia)CIS 2012), Tokyo, Japan, 9–10 August 2012. [[CrossRef](#)]
24. Arp, D.; Spreitzenbarth, M.; Hübner, M.; Gascon, H.; Rieck, K. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. In Proceedings of the 2014 Network and Distributed System Security Symposium, San Diego, CA, USA, 23–26 February 2014. [[CrossRef](#)]
25. Yerima, S.Y.; Sezer, S.; Muttik, I. High accuracy android malware detection using ensemble learning. *IET Inf. Secur.* **2015**, *9*, 313–320. [[CrossRef](#)]
26. Wang, X.; Wang, W.; He, Y.; Liu, J.; Han, Z.; Zhang, X. Characterizing Android apps' behavior for effective detection of malapps at large scale. *Future Gener. Comput. Syst.* **2017**, *75*, 30–45. [[CrossRef](#)]
27. Varsha, M.V.; Vinod, P.; Dhanya, K.A. Identification of malicious android app using manifest and opcode features. *J. Comput. Virol. Hacking Tech.* **2017**, *13*, 125–138. [[CrossRef](#)]
28. Fan, M.; Liu, J.; Wang, W.; Li, H.; Tian, Z.; Liu, T. DAPASA: Detecting Android Piggybacked Apps Through Sensitive Subgraph Analysis. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 1772–1785. [[CrossRef](#)]
29. Ban, T.; Takahashi, T.; Guo, S.; Inoue, D.; Nakao, K. Integration of Multi-modal Features for Android Malware Detection Using Linear SVM. In Proceedings of the 11th Asia Joint Conference on Information Security, Asia)CIS 2016, Fukuoka, Japan, 4–5 August 2016; pp. 141–146. [[CrossRef](#)]
30. Idrees, F.; Rajarajan, M. Investigating the android intents and permissions for malware detection. In Proceedings of the International Conference on Wireless and Mobile Computing, Networking and Communications, Larnaca, Cyprus, 8–10 October 2014; pp. 354–358. [[CrossRef](#)]
31. Kang, B.; Yerima, S.Y.; Sezer, S.; Mclaughlin, K. N-gram Opcode Analysis for Android Malware Detection. *Int. J. Cyber Situat. Aware.* **2016**, *1*, 231–254. [[CrossRef](#)]
32. Khan, R.U.; Zhang, X.; Kumar, R. Analysis of ResNet and GoogleNet models for malware detection. *J. Comput. Virol. Hacking Tech.* **2018**, 1–9. [[CrossRef](#)]
33. Kumar, R.; Xiaosong, Z.; Khan, R.U.; Kumar, J.; Ahad, I. Effective and Explainable Detection of Android Malware Based on Machine Learning Algorithms. In Proceedings of the 2018 International Conference on Computing and Artificial Intelligence, Chengdu, China, 12–14 March 2018; pp. 35–40.
34. Westyarian; Rosmansyah, Y.; Dabarsyah, B. Malware detection on Android smartphones using API class and machine learning. In Proceedings of the 5th International Conference on Electrical Engineering and Informatics: Bridging the Knowledge between Academic, Industry, and Community (ICEEI 2015), Denpasar, Indonesia, 10–11 August 2015; pp. 294–297. [[CrossRef](#)]
35. Wu, S.; Wang, P.; Li, X.; Zhang, Y. Effective detection of android malware based on the usage of data flow APIs and machine learning. *Inf. Softw. Technol.* **2016**, *75*, 17–25. [[CrossRef](#)]
36. Aung, Z.; Zaw, W. Permission-Based Android Malware Detection. *Int. J. Sci. Technol. Res.* **2013**, *2*, 228–234.
37. Peng, H.; Gates, C.; Sarma, B.; Li, N.; Qi, Y.; Potharaju, R.; Nita-Rotaru, C.; Molloy, I. Using probabilistic generative models for ranking risks of Android apps. In Proceedings of the 2012 ACM Conference on Computer And Communications Security—CCS '12, Raleigh, NC, USA, 16–18 October 2012. [[CrossRef](#)]
38. Pehlivan, U.; Baltaci, N.; Acarturk, C.; Baykal, N. The analysis of feature selection methods and classification algorithms in permission based Android malware detection. In Proceedings of the IEEE SSCI 2014: 2014 IEEE Symposium Series on Computational Intelligence—CICS 2014: 2014 IEEE Symposium on Computational Intelligence in Cyber Security, Orlando, FL, USA, 9–12 December 2014. [[CrossRef](#)]
39. Aafer, Y.; Du, W.; Droidapiminer: Mining Api-Level Features for Robust Malware Detection in Android. In Proceedings of the International Conference on Security and Privacy in Communication Systems, Sydney, NSW, Australia, 25–28 September 2013.
40. Itoh, Y.; Orlosky, J.; Huber, M.; Kiyokawa, K.; Klinker, G. OST Rift: Temporally consistent augmented reality with a consumer optical see-through head-mounted display. In Proceedings of the 2016 IEEE Virtual Reality (VR), Greenville, SC, USA, 19–23 March 2016. [[CrossRef](#)]
41. Chuang, H.Y.; Wang, S.D. Machine Learning Based Hybrid Behavior Models for Android Malware Analysis. In Proceedings of the 2015 IEEE International Conference on Software Quality, Reliability and Security, QRS 2015, Vancouver, BC, Canada, 3–5 August 2015. [[CrossRef](#)]
42. Xu, K.; Li, Y.; Deng, R.H. ICCDetector: ICC-Based Malware Detection on Android. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 1252–1264. [[CrossRef](#)]



43. Zhao, M.; Ge, F.; Zhang, T.; Yuan, Z. AntiMalDroid: An efficient SVM-based malware detection framework for android. *Commun. Comput. Inf. Sci.* **2011**, *243*, 158–166. [\[CrossRef\]](#)
44. Wu, W.C.; Hung, S.H. DroidDolphin: A dynamic android malware detection framework using big data and machine learning. In Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems, RACS 2014, Towson, MD, USA, 5–8 October 2014; pp. 247–252. [\[CrossRef\]](#)
45. Afonso, V.M.; de Amorim, M.F.; Grégio, A.R.A.; Junquera, G.B.; de Geus, P.L. Identifying Android malware using dynamically obtained features. *J. Comput. Virol. Hacking Tech.* **2015**, *11*, 9–17. [\[CrossRef\]](#)
46. Isohara, T.; Takemori, K.; Kubota, A. Kernel-based behavior analysis for android malware detection. In Proceedings of the 2011 7th International Conference on Computational Intelligence and Security (CIS 2011), Hainan, China, 3–4 December 2011. [\[CrossRef\]](#)
47. Ham, Y.J.; Lee, H.W. Detection of Malicious Android Mobile Applications Based on Aggregated System Call Events. *Int. J. Comput. Commun. Eng.* **2014**, *3*, 149. [\[CrossRef\]](#)
48. Ham, Y.J.; Moon, D.; Lee, H.W.; Lim, J.D.; Kim, J.N. Android mobile application system call event pattern analysis for determination of malicious attack. *Int. J. Secur. Its Appl.* **2014**, *8*, 231–246. [\[CrossRef\]](#)
49. Huda, S.; Islam, R.; Abawajy, J.; Yearwood, J.; Hassan, M.M.; Fortino, G. A hybrid-multi filter-wrapper framework to identify run-time behaviour for fast malware detection. *Future Gener. Comput. Syst.* **2018**, *83*, 193–207. [\[CrossRef\]](#)
50. Ferrante, A.; Malek, M.; Martinelli, F.; Mercaldo, F.; Milosevic, J. Extinguishing ransomware—A hybrid approach to android ransomware detection. In Proceedings of the International Symposium on Foundations and Practice of Security, Nancy, France, 23–25 October 2017; Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics); Springer: Berlin, Germany, 2018. [\[CrossRef\]](#)
51. Liu, Y.; Zhang, Y.; Li, H.; Chen, X. A hybrid malware detecting scheme for mobile Android applications. 2016 IEEE International Conference on Consumer Electronics (ICCE 2016), Las Vegas, NV, USA, 7–11 January 2016. [\[CrossRef\]](#)
52. Saracino, A.; Sgandurra, D.; Dini, G.; Martinelli, F. MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Trans. Dependable Secur. Comput.* **2018**, *15*, 83–97. TDSC.2016.2536605. [\[CrossRef\]](#)
53. Piatetsky-Shapiro, G. Discovery, analysis and presentation of strong rules. *Knowl. Discov. Databases* **1991**, 229–238. [\[CrossRef\]](#)



© 2019 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).